

1

2



Deleted:  
UDDI Spec TC

## UDDI Spec TC

3

# Schema Centric XML Canonicalization Version 1.0

4

5

### Document identifier:

SchemaCentricCanonicalization

6

7

### This version:

<http://uddi.org/pubs/SchemaCentricCanonicalization-20050523.htm>

8

9

10

### Latest version:

<http://uddi.org/pubs/SchemaCentricCanonicalization.htm>

11

12

### Previous version:

<http://uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm>

13

14

15

### Author:

Bob Atkinson, Microsoft, [bobatk@microsoft.com](mailto:bobatk@microsoft.com)

16

17

### Editors:

Selim Aissi, Intel, [selim.aisi@intel.com](mailto:selim.aisi@intel.com)  
Maryann Hondo, IBM, [mhondo@us.ibm.com](mailto:mhondo@us.ibm.com)  
[Andrew Hately, IBM, hately@us.ibm.com](mailto:Andrew.Hately@us.ibm.com)

18

19

20

21

### Abstract:

Existing XML canonicalization algorithms such as [Canonical XML](#) and [Exclusive XML Canonicalization](#) suffer from several limitations and design artifacts (enumerated herein) which significantly limit their utility in many XML applications, particularly those which validate and process XML data according to the rules of and flexibilities afforded by [XML Schema](#). The Schema Centric Canonicalization algorithm addresses these concerns.

22

23

24

25

26

27

28

29

30

### Status:

This specification has attained the status of Committee Specification. This document is updated periodically on no particular schedule.

31

32

33

34

Deleted: 20020710

35  
36  
37  
38  
39 |  
40 |  
41 |  
42

---

Committee members should send comments on this Committee Specification to the [uddi-spec@lists.oasis-open.org](mailto:uddi-spec@lists.oasis-open.org) list. Others should subscribe to and send comments to the [uddi-spec-comment@lists.oasis-open.org](mailto:uddi-spec-comment@lists.oasis-open.org) list. To subscribe, send an email message to <mailto:uddi-spec-comment-request@lists.oasis-open.org?body=subscribe> with the word "subscribe" as the body of the message.

43  
44  
45  
46  
47  
48

For information on whether any intellectual property claims have been disclosed that may be essential to implementing this Committee Specification, and any offers of licensing terms, please refer to the Intellectual Property Rights section of the UDDI Spec TC web page (<http://www.oasis-open.org/committees/uddi-spec/>).

49 |  
50

**Copyrights:**

51  
52  
53  
54  
55  
56

*Copyright © 2000-2002 by Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., VeriSign, Inc., and / or Microsoft Corporation. All Rights Reserved. See also [Appendix A: Notices](#).*

57  
58

*Copyright © OASIS Open 2002-2003. All Rights Reserved. See also [Appendix A: Notices](#).*

59

## Table of Contents

- 60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76
- 1. Introduction
    - 1.1 Limitations of Existing Canonicalization Algorithms
    - 1.2 Canonicalization Algorithms & Web Services Applications
  - 2. Overview of Schema Centric Canonicalization
    - 2.1 Algorithm Input
    - 2.2 Character Model Normalization
    - 2.3 Processing by XML Schema-Assessment
    - 2.4 Additional Infoset Transformation
    - 2.5 Serialization of the Schema-Canonicalized Infoset
    - 2.6 Limitations
  - 3. Specification of Schema Centric Canonicalization
    - 3.1 Creation of Input as an Infoset
      - 3.1.1 Conversion of an Octet Stream to an Infoset



117  
118  
119  
120

1. The presence of a DTD that validates the XML subdocument being canonicalized is [assumed](#). In particular, default attributes specified in the DTD are included in the output of the canonicalization process.

121  
122  
123  
124  
125  
126  
127  
128  
129

With the advent of [XML Schema](#), it is in fact now increasingly rare to find XML documents for which validation is accomplished using a DTD, or, indeed, due to the weak expressiveness of DTDs, to find XML documents for which a DTD which describes the content models of the elements of the document (instead of merely defining entities and the like) can in fact ever be constructed. Thus, the existing algorithms are becoming less and less useful to practical applications of XML.

130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159

2. Contrary to the [intent](#) of the Namespaces in XML Recommendation, XML documents are *not* canonicalized with respect to the XML namespace [prefixes](#) they use. That is, XML documents that are identical except for their choice of namespace prefixes canonicalize to different results under the existing algorithms. Since namespace declarations can appear on *any* element, the need for their preservation can at times be a very significant implementation burden.
3. Canonical XML contains a (pragmatically minor) security hole having to do with how it processes certain esoteric node-sets. Consider a node set which consists of just a single attribute node, one that explicitly references a namespace by use of a namespace prefix. While it is true in Canonical XML that an element node that is not in the node-set still has its namespace axis processed, the rule in Canonical XML (see [§2.3](#)) for processing that namespace axis states that only "namespace nodes in the axis *and in the node-set*" (*emphasis added*) are in fact processed. Thus, the canonical representation of our single-attribute-node node-set consists of the processing of *only* the attribute node itself; no namespace attributes are included. Thus, two such single-attribute node-sets whose attributes are character-wise identical but use completely different namespaces as the binding of their prefix will canonicalize to the same result, and that presents a security hole, particularly in applications to digital signatures. Analogous security holes exist with similar node-sets. Whether the same security hole exists in Exclusive XML Canonicalization is likely the case but is not entirely clear.

160  
161  
162  
163  
164  
  
165  
166  
167  
  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203

4. The [goal](#) of the existing canonicalization algorithms is to canonicalize an XML subdocument with respect to the liberties of its physical representation permitted within *only* the [XML 1.0 Recommendation](#) and the [Namespaces in XML Recommendation](#).

The XML Schema Recommendation permits a considerable number of additional liberties of representation, including (but not limited to) the following:

- a. the [optional presence](#) of both comments and processing instructions at completely arbitrary points in the input XML
- b. [normalization of whitespace](#) in certain element content (in a like manner as but in addition to the normalization of whitespace within *attributes* mandated by XML 1.0)
- c. the permitted presence of whitespace with no semantic impact imparted by the presence thereof in the content of elements of complex type which have a {content type} of *element-only* (that is, between end-tags and start-tags of elements which are children of such elements)
- d. the specification in a schema of the [default value](#) of attributes, which consequently permits without semantic impact their omission in a corresponding XML instance
- e. the specification in a schema of the [default value](#) of the content of elements, which operates in a manner similar to that of the specification of the default value of attributes
- f. the inclusion of [xsi:schemaLocation](#) and [xsi:noNamespaceSchemaLocation](#) attributes as useful hints to the XML Schema processing system but which are not of semantic significance to XML Schema instance itself
- g. within the content of an element of complex type which has a {content type} of *element-only*, the semantic insensitivity to the order within a sequence of elements that validates against a [model group](#) whose compositor is *all*. (In contrast, when such occurs within a {content type} of *mixed*, and so there may be non-whitespace interspersed between these elements, the elements may not reasonably be reordered, as their relationship to such characters may have semantic significance to applications.)

- 204 h. variability in the lexical representation of the data  
205 types [built-in to XML Schema](#) and extensions or  
206 restrictions thereof, including  
207 i. the permitted use of any of {true, false, 0, 1} for  
208 data of type [boolean](#)  
209 ii. the optional use of leading "+" signs in positive  
210 values, and the optional use of leading and  
211 trailing zeros in data of type [decimal](#) and  
212 restrictions thereof, including [integer](#), [long](#), [int](#),  
213 [nonNegativeInteger](#), and so on (as well as, of  
214 course, user-defined extensions and  
215 restrictions)  
216 iii. for data of type [float](#) and [double](#), the use of upper  
217 or lower case "e" in scientific notation, the use  
218 of leading zeros in the exponent thereof, the  
219 use of leading "+" signs on positive values, the  
220 use of trailing zeros in the mantissa, and the  
221 unnecessary use of leading zeros in the  
222 mantissa.  
223 iv. the permitted use of various time zones to  
224 represent the same time value in data of type  
225 [dateTime](#) and [time](#), as well as two  
226 representations for midnight for such data  
227 v. the permitted use of both upper and lower case in  
228 data of type [hexBinary](#)  
229 vi. in data of type [base64Binary](#), the permitted use  
230 (per the [clarification](#) in the [errata](#) to XML  
231 Schema of the lexical forms of [base64Binary](#)  
232 data) of whitespace characters

233 It should be noted that for these six data types, XML  
234 Schema Datatypes does in fact [normatively define](#) a  
235 corresponding canonical lexical representation. For  
236 example, the [canonical lexical representation of](#)  
237 [boolean](#) permits only the use of values in the set  
238 {true, false}. However, XML Schema makes use of  
239 this canonicalization only in certain circumstances,  
240 such as the interpretation of default values of  
241 attributes and elements.

242 There are further data type canonicalization issues  
243 which appear to have been overlooked by XML  
244 Schema Datatypes:

- 245 vii. (minor) It is not precisely clear from the XML  
246 Schema Datatypes specification whether

247 leading zeros are permitted in instances of  
248 [gYearMonth](#) and [gYear](#) when (the absolute  
249 value of) the year in question is outside the  
250 range of 0001 to 9999. However, in the  
251 otherwise analogous [passage](#) of the  
252 specification of `dateTime`, such ambiguity is  
253 not present (such leading zeros are  
254 prohibited), and a reasonable interpretation in  
255 these other two cases is to straightforwardly  
256 follow that precedent.

viii. the use of mixed case language-tags in data of  
257 type [language](#); this is permitted per section "2.  
258 The language tag" of [RFC 1766](#), which is  
259 (ultimately) the referenced normative  
260 specification for the value space of [language](#).  
261 (**Note:** this same value space [is used](#) by the  
262 `xml:lang` attribute as defined by the [XML 1.0](#)  
263 Recommendation; thus, the omission of the  
264 canonicalization of the case of `xml:lang`  
265 attributes should reasonably be considered a  
266 flaw in even the existing canonicalization  
267 algorithms.)

ix. More generally, it is often the case in real-world  
269 schemas that various string-valued attributes  
270 and elements defined therein are interpreted  
271 at the application level as being case-  
272 insensitive. This should be capable of being  
273 captured by the canonicalization algorithm;  
274 were it not, then applications may be forced to  
275 remember the exact case used for certain  
276 data, a requirement in tension with the  
277 application semantic, and quite possibly thus a  
278 significant implementation burden.  
279

## 280 **1.2 Canonicalization Algorithms & Web** 281 **Services Applications**

282 That these limitations are indeed considerably problematic can be  
283 more readily appreciated by considering the implications to certain  
284 types of application. One increasingly common and important  
285 application of XML is that of so-called "web services". For our  
286 purposes here, web services can be thought of as networked  
287 applications where the payloads conveyed between network  
288 nodes are XML documents, often [SOAP](#) requests or responses,  
289 which in turn have XML subdocuments in their [headers](#) and [body](#).  
290 It is observed to be the case that, almost universally, the

291 specification of what constitutes correct and appropriate XML in  
292 such circumstances is accomplished using XML Schema.

293 On the server side of web service applications, it is very often the  
294 case that the semantic information conveyed by a request needs  
295 to be decomposed, analyzed, and persistently stored, often  
296 making use of an underlying relational database to do so. To the  
297 extent that such a database is used for storage and indexing  
298 purposes, this database gets populated from data received in the  
299 body of XML "update" requests. Such population is carried out by  
300 "shredding" the semantic information of the XML into a  
301 corresponding representation in relational form, losing thereafter  
302 the history of that information as having originated in an XML form.  
303 Conversely, XML "get" requests are serviced by performing  
304 relational operations against the database, then forming an  
305 appropriate XML response based on the retrieved data and the  
306 schema to which the response must conform.

307 Certain web service applications will wish to support the use of  
308 digital signatures on content which is manipulated by the  
309 application. In order to reasonably support such usage, and, in  
310 particular, in order to continue to reasonably allow for the  
311 shredding of data into an underlying relational store, the signatures  
312 in question need to be canonicalized with respect to the full range  
313 of liberties of representation afforded by XML Schema. In  
314 particular, the problems with the existing algorithms enumerated in  
315 the previous section cause especially difficult implementation  
316 conundrums in these situations.

317 The Schema Centric Canonicalization Algorithm is intended to  
318 address these concerns.

## 319 **2. Overview of Schema Centric** 320 **Canonicalization**

321 The Schema Centric Canonicalization algorithm is intended to be  
322 complementary in a hand-in-glove manner to the processing of  
323 XML documents as carried out by the [assessment](#) of schema  
324 validity by XML Schema, canonicalizing its input XML instance with  
325 respect to *all* those representational liberties which are permitted  
326 thereunder. Moreover, the specification of Schema Centric  
327 Canonicalization heavily exploits the details and specification of  
328 the XML Schema validity-assessment algorithm itself.

329 In XML Schema, the analysis of an XML instance document  
330 requires that the document be modeled at the abstract level of an  
331 information set as defined in the [XML Information Set](#)



332 recommendation. Briefly, an XML document's information set  
333 consists of a number of **information items** connected in a graph.  
334 An information item is an abstract description of some part of an  
335 XML document: each information item has a set of associated  
336 named **properties**. By tradition, infoset property names are  
337 denoted in square brackets, [thus]. There are [eleven](#) different  
338 types of information items:

- 339 1. element information items,
- 340 2. attribute information items,
- 341 3. comment information items,
- 342 4. namespace information items,
- 343 5. character information items,
- 344 6. document information items,
- 345 7. processing instruction information items,
- 346 8. unexpanded entity reference information items,
- 347 9. document type declaration information items,
- 348 10. unparsed entity information items, and
- 349 11. notation information items.

350 Properties on each of these items, for example the [children]  
351 property of element information items, connect together items of  
352 different types in an intuitive and straightforward way.

353 The representation of an XML document as an *infoset* lies in  
354 contrast to its representation as a *node-set* as [defined](#) in *XPath*.  
355 The two notions are conceptually quite similar, but they are not  
356 isomorphic. For a given node-set it is possible to construct a  
357 semantically equivalent infoset without loss of information;  
358 however, the converse is not generally possible. It is the infoset  
359 abstraction which is the foundation of XML Schema, and it is  
360 therefore the infoset abstraction we use here as the foundation on  
361 which to construct Schema Centric Canonicalization algorithm.

362 The Schema Centric Canonicalization algorithm consists of a  
363 series of steps: creation of the input as an infoset, character model  
364 normalization, processing by XML-Schema assessment, additional  
365 infoset transformation, and serialization.

## 366 **2.1 Algorithm Input**

367 As was mentioned, the algorithm requires that the data it is to  
368 process be manifest as an infoset. If such is not provided directly  
369 as input, the data provided must be converted thereto. Two  
370 mechanisms for carrying out this conversion are defined:

- 371  
372  
373  
374  
375  
376  
377
1. If an octet stream is provided, then it is to be converted into an infoSet according to the definition in [\[XML-InfoSet\]](#) of the information set which results from the parsing of an XML document represented as an octet stream.
  2. If an XPath node-set is provided, then it is to be converted into an infoSet according to the [rules](#) defined below in this specification.

378  
379  
380  
381  
382  
383  
384

In addition to the data itself, the canonicalization process requires the availability of appropriate [XML Schemas](#) and an indication of the relevant components thereof to which the data purports to conform. In order to be able to successfully execute the canonicalization algorithm, all the data must be [valid](#) with respect to these components; data which is not valid cannot be canonicalized.

## 385 **2.2 Character Model Normalization**

386  
387  
388  
389  
390  
391

The [Unicode Standard](#) allows diverse representations of certain "precomposed characters" (a simple example is "ç"). Thus two XML documents with content that is equivalent for the purposes of most applications may contain differing character sequences. However, a normalized form of such representations is also [defined](#) by the Unicode Standard.

392  
393  
394  
395  
396  
397

Schema Centric Canonicalization requires that both its input infoSet and all the [schema components](#) processed by the XML Schema-Assessment process be transformed as necessary so that all string-valued properties and all sequences of character information items therein be normalized into the Unicode [Normalization Form C](#).

## 398 **2.3 Processing by XML Schema-** 399 **Assessment**

400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411

The third step of the Schema Centric Canonicalization requires that the input infoSet be transformed into the so-called "post-schema-validation infoSet" (the "PSVI") in the manner [defined](#) by the XML Schema Structures recommendation, amended as set forth below. In XML Schema, as the schema assessment process is carried out, the input infoSet is augmented by the addition of new properties which record in the information items various pieces of knowledge which the assessment process has been able to infer. For example, attribute information items are augmented with a [\[schema normalized value\]](#) property which contains the result of, among other things, the [application](#) of the appropriate schema-specified default-value to the attribute information item

412 (the full list of such augmentation is [tabulated](#) in the appendix to  
413 XML Schema Structures).

## 414 **2.4 Additional Infoset Transformation**

415 The PSVI output from XML Schema is next further transformed  
416 into what we define here as the "schema-canonicalized infoset" by  
417 rules of this specification that are designed to address a few  
418 remaining canonicalization issues:

- 419 1. the existence of information items in the info set which are  
420 completely ignored by the schema assessment process.
- 421 2. the existence of the semantically important use of XML  
422 namespace prefixes in various embedded languages which  
423 are contained strings of the input. For example, an attribute  
424 might in fact represent an XPath expression which may  
425 internally refer to contextual namespace prefixes. This issue  
426 is [discussed](#) at some length in Canonical XML. In that  
427 specification a decision was made to not canonicalize with  
428 respect to namespace prefixes due to the existence of such  
429 embedded languages, leaving the output of the algorithm  
430 sensitive to the particular prefixes used in the input. Here  
431 we choose otherwise, and provide a means by which the  
432 algorithm is desensitized to the use of namespace prefixes  
433 in embedded languages.
- 434 3. the namespaces which, in fact, are used in the output need  
435 to be canonicalized with respect to the namespace prefix  
436 declaration used for a given such namespace. The overall  
437 result is that the output of the Schema Centric  
438 Canonicalization algorithm is in no way sensitive to the  
439 particular choice of namespace prefixes in its inputs.
- 440 4. the [previously-mentioned](#) permitted variability in the  
441 representation of simple data types in XML Schema

## 442 **2.5 Serialization of the Schema- 443 Canonicalized Infoset**

444 Finally, the schema-canonicalized infoset is serialized into an XML  
445 text representation in a canonical manner, and this serialization  
446 forms the output of the algorithm.

447 The output of the Schema Centric Canonicalization algorithm  
448 whose input is the infoset of an entire XML document is well-  
449 formed XML. However, if some items in the infoset are logically  
450 omitted (that is, their [omitted](#) property is true), then the output  
451 may or may not be well-formed XML, depending on exactly which  
452 items are omitted (consider, for example, omitting some element

453 information items but retaining their attributes). However, since the  
454 canonical form may be subject to further XML processing, most  
455 infosets provided for canonicalization will be designed to produce  
456 a canonical form that is a well-formed XML document or external  
457 general parsed entity. Note that the Schema Centric  
458 Canonicalization algorithm [shares](#) these issues of well-formedness  
459 of output with the existing canonicalization algorithms.

460 In such cases where the output of the Schema Centric  
461 Canonicalization algorithm is well-formed, then the  
462 canonicalization process is idempotent: if  $x$  is the input infoset, and  
463  $C$  represents the application of the Schema Centric  
464 Canonicalization algorithm, then  $C(x)$  is identical to  $C(C(x))$ .  
465 Moreover, in such cases  $C(x)$  is [valid](#) with respect to the same  
466 schema component(s) as is  $x$  (modulo the character sequence  
467 length issue noted in the next section).

## 468 **2.6 Limitations**

469 The Schema Centric Canonicalization algorithm suffers from some  
470 of the [limitations](#) of Canonical XML. Specifically, as in Canonical  
471 XML, the [base URI] of the infoset has no representation in the  
472 canonicalized representation, the consequences of which are as in  
473 Canonical XML. However, unlike Canonical XML, Schema Centric  
474 Canonicalization does not suffer from the loss of notations and  
475 external unparsed entity references (these are canonicalized and  
476 preserved) nor from the loss of the typing of data (since, in XML  
477 Schema, the association of a schema with an XML instance is  
478 outside the scope of the specification and therefore is (trivially)  
479 preserved by the Schema Centric Canonicalization algorithm).

480 [As in](#) Exclusive XML Canonicalization, the XML being  
481 canonicalized may semantically depend on the effect of xml  
482 namespace attributes, such as [xml:lang](#) and [xml:space](#). As was  
483 the case in Exclusive XML Canonicalization, to avoid problems  
484 due to the importation of such attributes from information items  
485 which are omitted from the canonicalized output, either they must  
486 be explicitly given in the apex nodes of the XML information items  
487 being canonicalized or they must always be declared with an  
488 equivalent value in every context in which the XML information  
489 items will be interpreted.

490 Schema Centric Canonicalization **REQUIRES** the identification  
491 and availability of a relevant schema for the information items  
492 which are to be canonicalized. Therefore, information items which  
493 lack such schema cannot be canonicalized with this algorithm.

494 Schema Centric Canonicalization suffers (but arguably in a minor  
495 way) from the fact that XML schema-assessment is not strictly  
496 speaking deterministic: when an element or attribute information  
497 item is validated against a [wildcard](#) whose {process contents}  
498 property is *lax*, the exact schema-assessment processing of the  
499 item which takes place depends on whether "the item, or any items  
500 among its [\[children\]](#) if it's an element information item, has a  
501 uniquely determined declaration available", where the term  
502 "available" here provides a degree of discretion to the validating  
503 application and thus a degree of non-determinism to the schema-  
504 assessment process. Because Schema Centric Canonicalization  
505 makes integral use of the information garnered during schema-  
506 assessment, if an item has been *skipped* due to a wildcard with a  
507 {process contents} of *lax* or *skip*, the output of the algorithm for  
508 that item must necessarily be different than if the item has not  
509 been *skipped*. Thus, the non-determinism caused by *lax* results  
510 directly in non-determinism of the output of the algorithm. In order  
511 to reduce the actual occurrence of this non-determinism, to the  
512 extent that it does not conflict with other design requirements, it is  
513 RECOMMENDED that schemas intended for use with  
514 canonicalization avoid the use of a {process contents} of *lax* in  
515 their definitions.

516 In the canonicalized form, the lengths of certain sequences of  
517 character information items may differ from that which was input to  
518 the algorithm, due to both processing by Unicode character model  
519 normalization and to namespace attribute normalization (the latter  
520 only occurs for expressions written in embedded languages such  
521 as XPath). This length adjustment can in certain circumstances  
522 affect the validity of the altered data, and can affect the ability to  
523 reference into the data with [XPointer](#) character-points and ranges.

### 524 **3. Specification of Schema Centric** 525 **Canonicalization**

526 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL",  
527 "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED",  
528 "MAY", and "OPTIONAL" in this document are to be interpreted as  
529 described in [RFC 2119](#).

530 The specification of the Schema Centric Canonicalization  
531 algorithm defines a few items residing in an XML namespace  
532 known as the Schema Centric Canonicalization algorithm  
533 namespace. The URI of this namespace is:

534 urn:uddi-  
535 org:SchemaCentricC14N:2002-07-10

536 A (non-normative) XML Schema .xsd file containing definitions of  
537 the members of this namespace defined by this specification can  
538 be found at:

539 [http://www.uddi.org/schema/SchemaCe](http://www.uddi.org/schema/SchemaCentricCanonicalization.xsd)  
540 [ntricCanonicalization.xsd](http://www.uddi.org/schema/SchemaCentricCanonicalization.xsd)

541 It should be clearly understood that all the details of the present  
542 document are a matter solely of the *specification* of the behavior of  
543 the Schema Centric Canonicalization algorithm, not its  
544 *implementation*. Implementations are (of course) free to pursue  
545 any course of implementation they choose so long as in all cases  
546 the output they yield for a given input corresponds exactly to that  
547 as is indicated herein. At times the details and language in this  
548 specification may have been optimized to attempt to make the  
549 presentation and specification more clear and straightforward  
550 perhaps at the performance expense of an implementation that  
551 were to robotically follow the literal wording thereof. In this regard,  
552 attention is specifically drawn to the connection of the this  
553 specification with the details of the specification of the XML  
554 Schema validity-assessment, the PSVI augmentation process, and  
555 the augmentation of the PSVI found in §3.3: depending on the  
556 existing software artifacts and other resources upon which they  
557 can rely, good implementations are likely to significantly optimize  
558 their treatment of these matters especially.

### 559 **3.1 Creation of Input as an Infoset**

560 The Schema Centric Canonicalization algorithm manipulates the  
561 semantic information of an XML instance as represented in the  
562 form of an [XML Information Set](#). As such, if the input to the  
563 algorithm is not already in this form then it must be converted  
564 thereto in order for the algorithm to proceed. This document  
565 normatively specifies the manner in which this conversion is to be  
566 carried out for two such alternative input data-types (other  
567 specifications are free to define additional, analogous  
568 conversions). These two data-types are exactly those [defined](#) by  
569 the XML Signature Syntax and Processing recommendation as  
570 being the architected data-types for input to a Transform.

571 As is [noted](#) in the XML Information Set recommendation, it is not  
572 intrinsically the case that the [in-scope namespaces] property of an  
573 element information item in an infoset will be consistent with the  
574 [namespace attributes] properties of the item and its ancestors,  
575 though this is always true for an information set resulting from

576 parsing an XML document. However, it is REQUIRED that this  
577 consistency relationship hold for the infoset input to the Schema  
578 Centric Canonicalization algorithm.

### 579 **3.1.1 Conversion of an Octet Stream to an** 580 **Infoset**

581 If the input to the canonicalization algorithm is an octet stream,  
582 then it is to be converted into an infoset by parsing the octet  
583 stream as an XML document in the manner described in the  
584 specification of [\[XML-Infoset\]](#).

585 Note that this is exactly the same conversion process that [must be](#)  
586 [carried out](#) by software attempting to assess the schema validity of  
587 XML data according to the XML Schema Structure  
588 recommendation.

### 589 **3.1.2 Conversion of a Node-set to an Infoset**

590 The conversion of a node-set to an infoset is straightforward, if  
591 somewhat more lengthy to describe.

592 A node-set is [defined](#) by the [XPath recommendation](#) as "an  
593 unordered collection of nodes without duplicates." In this context,  
594 the term "node" refers to the definition provided in the [data model](#)  
595 [section](#) of the recommendation. In that section, it is noted that  
596 XPath operates on an XML document as a tree, and that there are  
597 seven types of node that may appear in such trees:

- 598 1. root nodes
- 599 2. element nodes
- 600 3. attribute nodes
- 601 4. text nodes
- 602 5. namespace nodes
- 603 6. processing instruction nodes
- 604 7. comment nodes

605 The nodes in a given node-set must (by construction; that is, rules  
606 that would allow otherwise are lacking in XPath) all be nodes from  
607 the same underlying tree instance. If  $N$  is a node-set, then let  $T(N)$   
608 be this tree, and let  $r(T(N))$  be the root node of that tree. The  
609 conversion process to an infoset first converts  $T(N)$  into an  
610 equivalent infoset  $I(T(N))$ , then decorates that infoset to denote  
611 which information items therein correspond to nodes originally  
612 found in  $N$ .

613 Conversion of an XPath node-tree to an infoset is defined  
614 recursively in terms of the conversion of individual nodes to

615 corresponding information items. Let  $n$  be an arbitrary XPath node,  
616 and let  $\{n\}$  be a node-set containing just the node  $n$ . Let  $i$  be the  
617 function taking a node as input and returning an ordered list of  
618 nodes as output which is defined as follows:

619 1. If  $n$  is a root node, then  $i(n)$  is a single **document**  
620 **information item**, where:

621 a. the **[children]** property is the ordered list resulting  
622 from the concatenation of the lists of information  
623 items

624  $i(c_j)$

625 , where  $c_j$  ranges over the children of  $n$  in document  
626 order, excepting that those children of  $n$  (if any)  
627 contained within the DTD (if one exists; entity  
628 declarations, for example, may still usefully be found  
629 therein even if XML Schema is used for validation)  
630 are excluded.

631 b. the **[document element]** property is either  
632 i. that member of **[children]** which results from the  
633 conversion of the single child of  $n$  which is an  
634 element node, if such is present, or  
635 ii. **no value**, if such is not present.  
636 c. the **[notations]** property has **no value**.  
637 d. the **[unparsed entities]** property has **no value**.  
638 e. the **[base URI]** property is **unknown**.  
639 f. the **[character encoding scheme]** property is  
640 **unknown**.  
641 g. the **[standalone]** property has **no value**.  
642 h. the **[version]** property has **no value**.  
643 i. the **[all declarations processed]** property is false.

644 2. If  $n$  is an element node, then  $i(n)$  is a single **element**  
645 **information item**, where:

646 a. the **[namespace name]** property is the result of the  
647 function invocation  $namespace-uri(\{n\})$   
648 b. the **[local name]** property is the result of the function  
649 invocation  $local-name(\{n\})$   
650 c. the **[prefix]** property is either  
651 i. the prefix of the **QName** which is the result of the  
652 function invocation  $name(\{n\})$ , if such result is  
653 not the empty string, or  
654 ii. **no value** otherwise.



- 655 d. the **[children]** property is the ordered list resulting  
656 from the concatenation of the lists of information  
657 items
- 658  $i(c_i)$
- 659 , where  $c_i$  ranges over the children of  $n$  in document  
660 order
- 661 e. the **[attributes]** property is the unordered set whose  
662 members are the collective members of the lists of  
663 information items
- 664  $i(a_j)$
- 665 , where  $a_j$  ranges over those **attribute nodes** in  $T(\{n\})$   
666 whose **parent** is  $n$  (note that such attribute nodes **are**  
667 **not children** of  $n$ ).
- 668 f. the **[in-scope namespaces]** property is the  
669 unordered set whose members are the collective  
670 members of the lists of information items
- 671  $i(n_k)$
- 672 (which are by construction namespace information  
673 items), where  $n_k$  ranges over the set of **namespace**  
674 **nodes** in  $T(\{n\})$  whose **parent** is  $n$  (note such  
675 namespace nodes are not *children* of  $n$ ).
- 676 g. the **[namespace attributes]** property is an  
677 unordered set of attribute information items  
678 computed as follows. Let  $Nn$  be the set of  
679 namespace information items in the [in-scope  
680 namespaces] property of  $i(n)$ , and let  $Np$  be the set of  
681 namespace information items in the [in-scope  
682 namespaces] property of  $i(m)$ , where  $m$  is the  
683 [parent] of  $n$ . For each namespace information item  $s$   
684 in  $Nn - Np$  (so, each namespace information item  
685 newly introduced on  $i(n)$ ), the [namespace attributes]  
686 property contains an attribute information item whose  
687 properties are as follows:  
688 i. the [namespace name] property is (per **XML**  
689 **Infoset**) "<http://www.w3.org/2000/xmlns/>"  
690 ii. the [local name] property is the value of the  
691 [prefix] property of  $s$ .

- 692                   iii. the [prefix] property is "xmlns"  
693                   iv. the [normalized value] property is the value of the  
694                   [namespace name] property of *s*.  
695                   v. the remaining properties are as set forth in the  
696                   attribute node **case** below.

697                   Conversely, consider each namespace node *s* in  $N_p$   
698                   -  $N_n$  (so, each namespace information item present  
699                   on the parent but removed on *n*). The specification of  
700                   XML Namespaces is such that there can be at most  
701                   one such *s*, and that it represent a declaration of the  
702                   default namespace, which is then undeclared by  
703                   element *i(n)*. If such an *s* exists, then the  
704                   [namespace attributes] property of *i(n)* additionally  
705                   contains an attribute information item whose  
706                   properties are as follows:

- 707                   vi. the [namespace name] property is  
708                   "http://www.w3.org/2000/xmlns/"  
709                   vii. the [local name] property is the empty string  
710                   viii. the [prefix] property is "xmlns"  
711                   ix. the [normalized value] property is the empty  
712                   string  
713                   x. the remaining properties are as set forth in the  
714                   attribute node **case** below.  
715                   h. the [base URI] property is **unknown**.  
716                   i. the [parent] property is the document or element  
717                   information item in the infoset rooted at  $i(r(T(\{n\}))$   
718                   which contains this information item in its [children]  
719                   property.

- 720                   3. If *n* is an attribute node, then *i(n)* is a single **attribute**  
721                   **information item**, where:  
722                   a. the [namespace name] property is the result of the  
723                   function invocation  $namespace-uri(\{n\})$   
724                   b. the [local name] property is the result of the function  
725                   invocation  $local-name(\{n\})$   
726                   c. the [prefix] property is either  
727                   i. the prefix of the QName which is the result of the  
728                   function invocation  $name(\{n\})$ , if such result is  
729                   not the empty string, or  
730                   ii. **no value** otherwise.  
731                   d. the [normalized value] property is the result of the  
732                   function invocation  $string(\{n\})$   
733                   e. the [specified] property is **unknown**.  
734                   f. the [attribute type] property is **unknown**.  
735                   g. the [references] property is **unknown**.

736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780

- h. the **[owner element]** property is the element information item in the infoset rooted at  $i(r(T(\{n\}))$  which contains this information item in its [attributes] property, if any such element exists, or **no value** otherwise.
- 4. If  $n$  is a text node, then  $i(n)$  is an ordered list of **character information items**, one character information item  $c_j$  corresponding to each character in the result of the function invocation  $string(\{n\})$ , where
  - a. the **[character code]** property of  $c_j$  is the ISO 10646 character code of the corresponding  $j$ th character in the result of the function invocation  $string(\{n\})$ .
  - b. the **[element content whitespace]** property of  $c_j$  is
    - i. **unknown** if the character is **whitespace**, and
    - ii. **false** otherwise.
  - c. the **[parent]** property is the element information item in the infoset rooted at  $i(r(T(\{n\}))$  which contains this information item in its [children] property.
- 5. If  $n$  is a namespace node, then  $i(n)$  is a single **namespace information item**, where
  - a. the **[prefix]** property is the result of the function invocation  $local-name(\{n\})$ , unless that returns an empty string, in which case the [prefix] property is **no value**. This perhaps unexpected formulation arises from the **fact** that in XPath, "a namespace node has an **expanded-name**: the local part is the namespace prefix (this is empty if the namespace node is for the default namespace); the namespace URI is always null."
  - b. the **[namespace name]** property is the result of the function invocation  $string(\{n\})$ .
- 6. If  $n$  is a processing instruction node, then  $i(n)$  is a single **processing instruction information item**, where
  - a. the **[target]** property is the result of the function invocation  $local-name(\{n\})$ .
  - b. the **[content]** property is the result of the function invocation  $string(\{n\})$ .
  - c. the **[base URI]** property is **unknown**.
  - d. the **[notation]** property is **unknown**.
  - e. the **[parent]** property is the document, element, or document type definition information item in the infoset rooted at  $i(r(T(\{n\}))$  which contains this information item in its [children] property
- 7. If  $n$  is a comment node, then  $i(n)$  is a single **comment information item**, where

- 781 a. the **[content]** property is the result of the function  
782 invocation *string*(*n*).
- 783 b. the **[parent]** property is the document or element  
784 information item in the infoSet rooted at *i(r(T({n}))*)  
785 which contains this information item in its [children]  
786 property.

787 Having defined the function *i*, we now return to completing the  
788 specification of the details of the node-set to infoSet conversion  
789 process.

790 Let *N* be a node-set, and consider the document information item  
791 returned by the function invocation *i(r(T(N)))*. Define the infoSet  
792 *I(T(N))* to be that set of information items which are transitively  
793 reachable from *i(r(T(N)))* through any of the properties defined on  
794 any of the information items therein. This infoSet represents the  
795 conversion of the node tree *T(N)* into a corresponding infoSet.

796 Recall that the node-set *N* is in fact a subset of *T(N)*. This  
797 relationship therefore needs to be represented in *I(T(N))*. To that  
798 end, we here define a new boolean infoSet property called  
799 **[omitted]**. Unless otherwise indicated by some specification, the  
800 value of the [omitted] property of any information item is always to  
801 be taken to be 'false'. The present specification, however, defines  
802 that for all information items in *I(T(N))* the value of [omitted] is 'true'  
803 except those items which, for some *n* in *N*, are members of the list  
804 returned from *i(n)*.

805 This completes the specification of the node-set to infoSet  
806 conversion process.

## 807 **3.2 Character Model Normalization**

808 The [Unicode Standard](#) allows diverse representations of certain  
809 "precomposed characters" (a simple example is "ç"). Thus two  
810 XML documents with content that is equivalent for the purposes of  
811 most applications may contain differing character sequences.  
812 However, a normalized form of such representations is also  
813 [defined](#) by the Unicode Standard.

814 It is REQUIRED in Schema Centric Canonicalization that both the  
815 input infoSet provided thereto and all the [schema components](#) to  
816 processed by the XML Schema-Assessment process [used therein](#)  
817 be transformed as necessary so that all string-valued properties  
818 and all sequences of character information items therein be  
819 normalized into the Unicode [Normalization Form C](#) as specified by  
820 the algorithm defined by the Unicode Standard.

821 As a (non-normative) note of implementation, in the case where  
822 the to-be-canonicalized XML instance and the XML schema  
823 specifications thereof are input to the canonicalization process as  
824 physical files, this normalization can usually be most  
825 straightforwardly accomplished simply by normalizing the  
826 characters of these files first before commencing with the  
827 remainder of the canonicalization process.

### 828 **3.3 Processing by XML Schema** 829 **Assessment**

830 Once the input infoset is normalized with respect to its character  
831 model, the Schema Centric Canonicalization algorithm carries out  
832 schema assessment by appealing to the third approach listed in  
833 §5.2 [Assessing Schema-Validity](#) of the XML Schema  
834 recommendation and attempting to carry out [strict](#) assessment of  
835 the element information item which is the value of the [document  
836 element] property of the document information item of the infoset.

837 In XML Schema, as the schema assessment process is carried  
838 out, the infoset input to that process is augmented by the addition  
839 of new properties which record in the information items various  
840 pieces of knowledge which the assessment process has been able  
841 to discern. For example, attribute information items are augmented  
842 with a [\[schema normalized value\]](#) property which contains the  
843 result of, among other things, the [application](#) of the appropriate  
844 schema-specified default-value to the attribute information item.

845 The Schema Centric Canonicalization algorithm makes use of this  
846 augmentation. Specifically, suppose  $I$  is the character-normalized  
847 version of the infoset which is input to the algorithm, possibly after  
848 conversion from another data-type. Then the next step of the  
849 algorithm forms the so-called "[post-schema-validation infoset](#)" (the  
850 "PSVI", or more precisely,  $PSVI(I)$ ) in exactly the manner  
851 prescribed as a consequence of [the assessment](#) process defined  
852 in the XML Schema Structures specification as amended in the  
853 manner set forth below. If  $PSVI(I)$  cannot be so formed, due to, for  
854 example, a failure of validation, then the Schema Centric  
855 Canonicalization algorithm terminates with a fatal error.

856 In XML Schema Structures, the augmentation process of schema  
857 assessment fails to record a small number of pieces of information  
858 which it has learned and which we find crucially necessary to have  
859 knowledge of here. Accordingly, the PSVI generation process  
860 referred to by this specification is exactly that of the [XML Schema](#)  
861 [Structures](#) recommendation as amended as follows:

862  
863  
864  
865  
866  
867  
868  
869  
870  
871

### 3.8.5 Model Group Information Set Contributions

If the schema-validity of an element information item has been assessed as per [Element Sequence Valid \(§3.8.4\)](#) by a model group whose {compositor} is *all*, then in the post-schema-validation info set it has the following property:

872  
873  
874

<b>PSVI Contributions for element information items</b>
<code>[validating model group all]</code> An <code>·item isomorphic·</code> to the model group component involved in such assessment.

875  
876  
877  
878  
879  
880

881  
882  
883  
884  
885  
886

## 3.4 Additional Info set Transformation

The fourth step of the Schema Centric Canonicalization algorithm further augments and transforms the PSVI to produce the "schema-canonicalized info set". This involves a pruning step, a namespace prefix desensitization step, a namespace attribute normalization step, and a data-type canonicalization step.

887  
888  
889  
890  
891  
892  
893  
894  
895

### 3.4.1 Pruning

Some information items in the PSVI in fact do not actively participate in the schema assessment process of XML Schema. They are either ignored completely by that process, or used in an administrative capacity which is not central to the outcome. Thus, these items need to be pruned from the PSVI in order that they not affect the output of canonicalization. Similarly, declarations of notations and unparsed entities which are not actually referenced in the canonicalized representation should also be removed.

896  
897

To this end, the `[omitted]` property is set to 'true' for any information item *info* in the PSVI for which at least one of the following is true:

898  
899  
900

1. *info* is a (necessarily whitespace) character information item which is a member of the `[children]` of an element information item whose `[type definition]` is a complex type

901 schema component whose {content type} property is  
902 *element-only*  
903 2. *info* is an attribute information item whose [namespace  
904 name] is identical to "[http://www.w3.org/2001/XMLSchema-](http://www.w3.org/2001/XMLSchema-instance)  
905 [instance](http://www.w3.org/2001/XMLSchema-instance)" and whose [local name] is one of  
906 "schemaLocation" or "noNamespaceSchemaLocation"  
907 3. *info* is a notation information item for which there does not  
908 exist an attribute or element information item in the infoSet  
909 whose [omitted] property is false, whose [member type  
910 definition] (if present) or [type definition] (otherwise)  
911 property is either  
912 a. a **NOTATION** simple type (or restriction or extension  
913 thereof)  
914 b. a **list** of same

915 and whose [schema normalized value] is identical (in the  
916 former case) or contains a list item which is identical (in the  
917 later case) to the [name] of the notation information item

918 4. *info* is an unparsed entity information item for which there  
919 does not exist an attribute or element information item in the  
920 infoSet whose [omitted] property is false, whose [member  
921 type definition] (if present) or [type definition] (otherwise)  
922 property is either  
923 a. an **ENTITY** simple type (or restriction or extension  
924 thereof)  
925 b. a **list** of same

926 and whose [schema normalized value] is identical (in the  
927 former case) or contains a list item which is identical (in the  
928 later case) to the [name] of the unparsed entity information  
929 item

### 930 **3.4.2 Namespace Prefix Desensitization**

931 The goal of namespace prefix desensitization is to first identify  
932 those information items in the infoSet which make use of  
933 namespace prefixes outside of XML start and end tags (that is,  
934 information of type **QName** and derivations and lists thereof as  
935 well as information representing an expression written in some  
936 embedded language which makes use of the XML Namespaces  
937 specification in a embedded-language-specific manner), and next  
938 to annotate the infoSet in order to indicate exactly where and in  
939 what manner uses of particular XML namespace prefixes in fact  
940 occur. That is, desensitization is a two-step process: a data  
941 location step, followed by an annotation step.

942 Note that the notion of embedded language used here includes not  
943 only languages (such as XPath) which are represented in XML as  
944 the content of certain strings but also those (such as XML Query)  
945 which make use of *structured* element content. In all cases,  
946 however, in order to be namespace-desensitizeable it is  
947 REQUIRED that all references to XML namespace prefixes do in  
948 fact ultimately lie in information identified as being of a simple type  
949 (usually strings). It is, however, permitted that these prefixes may  
950 be found in simple types which are attributes and / or the content  
951 of elements perhaps deep in the sub-structure of the element  
952 rooting the occurrence of the embedded language.

953 Moreover, in order to be namespace-desensitizeable, it is  
954 REQUIRED that the semantics of each embedded language not  
955 be sensitive to the specific namespace prefixes used, or the  
956 character-count length thereof: one MUST be permitted to  
957 (consistently) rewrite any or all of the prefixes used in an  
958 occurrence of a language with arbitrary other (appropriately  
959 declared) prefixes, possibly of different length, without affecting the  
960 semantic meaning in question.

961 Each particular embedded language for which namespace  
962 desensitization is to be done MUST be identified by a name  
963 assigned to it by an appropriate authority. It is REQUIRED that this  
964 name be of data-type [anyURI](#). This specification assigns the  
965 following URIs as names of particular embedded languages:

**URI**

<http://www.w3.org/TR/1999/REC-xpath-19991116>

<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>

**Embedded Language**

the embedded language which consists of sequences of characters which conform to the any of the grammatical productions of the [XPath 1.0](#) specification

an embedded language which consists of sequences of characters which are of type [QName](#) or derivations and/or lists (and their derivations) thereof

966 The data location step of desensitization makes use of  
967 canonicalization-specific [annotations](#) to XML Schema  
968 components. It is the case in XML Schema that the XML  
969 representation of all schema components allows the presence of  
970 attributes qualified with namespace names other than the XML  
971 Schema namespace itself; this is manifest in the schema-for-  
972 schemas as the presence of an



973  
974

```
<xs:anyAttribute namespace="##other"  
processContents="lax"/>
```

975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985

definition in the schema for each of the various schema components. As is [specified](#) in XML Schema Structures, such attributes are represented in the infoset representation of the schema inside the {attributes} property of an Annotation schema component which in turn is the value of the {annotation} property of the annotated schema component in question (i.e.: the Annotation is the {annotation} of the [Attribute Declaration](#), the [Element Declaration](#), or whatever). Within the Schema Centric Canonicalization algorithm [namespace](#), we define a couple of attributes intended for use as such annotations to schema components:

986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016

1. The **embeddedLang** attribute, which is of type [anyURI](#), is defined in the Schema Centric Canonicalization algorithm [namespace](#). When used as an attribute annotation to a schema component, an embeddedLang attribute indicates that an information item which validates against the schema component in question in fact contains information written in a certain, fixed embedded language whose name is indicated in the value of the embeddedLang attribute. The embeddedLang attribute may also be used within a schema instance (but only, of course, where such attributes are permitted by the corresponding schema); this is in loose analogy to how the [xsi:type](#) attribute is used. In such situations, the [owner element] of the embeddedLang attribute in fact contains information written in a certain, fixed embedded language whose name is indicated in the value of the embeddedLang attribute. The use of an embeddedLang attribute in a schema instance supercedes any identification of embedded language that may be provided by its schema.
2. The **embeddedLangAttribute** attribute, which is of type [QName](#), is defined in the Schema Centric Canonicalization algorithm [namespace](#). When used as an attribute annotation to a schema component, an embeddedLangAttribute attribute indicates that an information item which validates against the schema component in question in fact contains information written in an embedded language whose name is dynamically indicated in the information item (necessarily an element information item) as the value of a certain attribute thereof, namely the attribute whose qualified name is indicated in the value of the embeddedLangAttribute attribute.

1017 In order to specify how these attributes are used, we define an  
1018 auxiliary function in order to model the inheritance of annotations  
1019 in schemas from types to elements and attributes and from base  
1020 types to derived types. Let *i* be an information item, *a* be a string  
1021 (representing the name of an attribute), and *ns* be either a URI  
1022 (representing the name of an XML namespace) or the value  
1023 [absent](#). Define the function *getAnnot*(*i*, *a*, *ns*) as follows:

- 1024 1. If *i* is an element information item, then
  - 1025 a. If the [\[element declaration\]](#) property of *i* contains in  
1026 its {annotation} property an Annotation schema  
1027 component which contains in its {attributes} property  
1028 an attribute information item whose {name} is *a* and  
1029 whose {target namespace} is *ns* (that is, if the  
1030 [\[element declaration\]](#) property of *i* "has an (a,ns)  
1031 annotation attribute"), then *getAnnot*(*i*, *a*, *ns*) is the  
1032 value of that attribute
  - 1033 b. Otherwise, let *t* be the [\[member type definition\]](#)  
1034 property of *i* (if it exists) or the [\[type definition\]](#)  
1035 property of *i* (otherwise). Then *getAnnot*(*i*, *a*, *ns*) is  
1036 *getAnnot*(*t*, *a*, *ns*).
- 1037 2. If *i* is an attribute information item, then
  - 1038 a. If the [\[attribute declaration\]](#) property of *i* has an (a,ns)  
1039 annotation attribute, then *getAnnot*(*i*, *a*, *ns*) is the  
1040 value of that attribute.
  - 1041 b. Otherwise, let *t* be the [\[member type definition\]](#)  
1042 property of *i* (if it exists) or the [\[type definition\]](#)  
1043 property of *i* (otherwise). Then *getAnnot*(*i*, *a*, *ns*) is  
1044 *getAnnot*(*t*, *a*, *ns*).
- 1045 3. If *i* is an information item which is [item isomorphic](#) to a  
1046 [complex type definition](#) schema component, then,
  - 1047 a. If *i* has an (a,ns) annotation attribute, then  
1048 *getAnnot*(*i*, *a*, *ns*) is the value of that attribute
  - 1049 b. If the {base type definition} property *t* of *i* is not the  
1050 [ur-type definition](#), then *getAnnot*(*i*, *a*, *ns*) is  
1051 *getAnnot*(*t*, *a*, *ns*)
  - 1052 c. Otherwise, *getAnnot*(*i*, *a*, *ns*) is [absent](#).
- 1053 4. If *i* is an information item which is [item isomorphic](#) to a  
1054 [simple type definition](#) schema component, then,
  - 1055 a. If *i* has an (a,ns) annotation attribute, then  
1056 *getAnnot*(*i*, *a*, *ns*) is the value of that attribute.
  - 1057 b. If the {variety} property of *i* is *atomic*, and if the {base  
1058 type definition} property *t* of *i* is not the [ur-type  
1059 definition](#), then *getAnnot*(*i*, *a*, *ns*) is *getAnnot*(*t*, *a*, *ns*)

- 1060                   c. If the {variety} property of *i* is *list*, then *getAnnot(i, a,*  
1061                    *ns)* is *getAnnot(t, a, ns)*, where *t* is the {item type  
1062                    definition} property of *i*.  
1063                   d. Otherwise, *getAnnot(i, a, ns)* is **absent**.  
1064                   5. Otherwise, *getAnnot(i, a, ns)* is **absent**.

1065                   The data location step of desensitization is carried out as follows.  
1066                   Let *sccns* be the Schema Centric Canonicalization namespace.  
1067                   Consider in turn each attribute and element information item *x* in  
1068                   the pruned PSVI:

- 1069                   1. If *x* is an element information item, and if the [attributes] of *x*  
1070                    contain an attribute *a* whose [namespace name] is *sccns*  
1071                    and whose [local name] is "*embeddedLang*", then *x* is  
1072                    identified as being associated with the embedded language  
1073                    which is the value of the [schema normalized value] of *a* (if  
1074                    present) or the [normalized value] of *a* (otherwise).  
1075                   2. Otherwise, if *x* is an element information item, and if  
1076                    *getAnnot(x, "embeddedLangAttribute", sccns)* is not **absent**,  
1077                    then *x* is identified as being associated with the embedded  
1078                    language which is the [schema normalized value] (if  
1079                    present) or the [normalized value] (otherwise) of the  
1080                    member of the [attributes] of *x* whose name is the value of  
1081                    *getAnnot(x, "embeddedLangAttribute", sccns)*; if no such  
1082                    member of [attributes] exists, a fatal error occurs.  
1083                   3. Otherwise, if *getAnnot(x, "embeddedLang", sccns)* is not  
1084                    **absent**, then *x* is identified as being associated with the  
1085                    embedded language which is the value thereof;  
1086                   4. Otherwise, *x* is not associated with any embedded  
1087                    language by means of the *embeddedLang* or  
1088                    *embeddedLangAttribute* attributes, though such an  
1089                    association may be indicated by other means, such as by  
1090                    *fiat* in some specification.

1091                   To that last point this specification REQUIRES that a schema  
1092                   component representing any of the following:

- 1093                   1. the type of the element named "XPath" contained in  
1094                    elements of type *dsig:TransformType* (where the prefix  
1095                    "*dsig*" is bound to the XML Signature Syntax and  
1096                    Processing namespace:  
1097                    <http://www.w3.org/2000/09/xmlsig#>), or  
1098                   2. the "*xpath*" attribute whose [owner element] is the element  
1099                    *xsd:selector* (where the prefix "*xsd*" is bound to the XML  
1100                    Schema namespace:  
1101                    <http://www.w3.org/2001/XMLSchema>), or

1102 3. the "xpath" attribute whose [owner element] is the element  
1103 xsd:field (where the prefix "xsd" is bound as before)

1104 is to be considered by definition as possessing an embeddedLang  
1105 attribute with value [http://www.w3.org/TR/1999/REC-xpath-](http://www.w3.org/TR/1999/REC-xpath-19991116)  
1106 [19991116](http://www.w3.org/TR/1999/REC-xpath-19991116) in the {attributes} property of its {annotation} property  
1107 (that is, they are by definition annotated as being XPath 1.0  
1108 expressions).  
1109 Moreover, any attribute or element information item whose  
1110 [member type definition] (if present) or [type definition] (otherwise)  
1111 property is any of:

- 1112 1. [QName](#) or a derivation thereof
- 1113 2. a [list](#) of [QName](#) or a derivation thereof
- 1114 3. a derivation of a [list](#) of [QName](#) or a derivation thereof

1115 is identified as being associated with the embedded language  
1116 whose name is [http://www.w3.org/TR/2001/REC-xmlschema-2-](http://www.w3.org/TR/2001/REC-xmlschema-2-20010502)  
1117 [20010502](http://www.w3.org/TR/2001/REC-xmlschema-2-20010502).

1118 Other specifications are encouraged to provide similar legacy-  
1119 supporting definitions when appropriate: the price of not identifying  
1120 an embedded language when one is actually in use is that the  
1121 canonicalized output will (almost certainly) be non-operational due  
1122 to dangling or erroneously-bound namespace prefixes.

1123 Following the data location step, the processing of the attribute  
1124 and element information items identified as being associated with  
1125 embedded languages is carried out by the annotation step of  
1126 namespace prefix desensitization in what is necessarily an  
1127 embedded-language-specific manner. Implementations of the  
1128 Schema Centric Canonicalization algorithm will need to  
1129 understand the syntax and perhaps some semantics of each of the  
1130 embedded languages whose uses they encounter as they carry  
1131 out canonicalization. Should an embedded language which is not  
1132 appropriately understood be encountered, the Schema Centric  
1133 Canonicalization algorithm terminates with a fatal error. All  
1134 implementations of the Schema Centric Canonicalization algorithm  
1135 MUST in this sense fully understand the (XPath) embedded  
1136 language identified as [http://www.w3.org/TR/1999/REC-xpath-](http://www.w3.org/TR/1999/REC-xpath-19991116)  
1137 [19991116](http://www.w3.org/TR/1999/REC-xpath-19991116) as well as the embedded language identified as  
1138 <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>.  
1139 In all cases the execution of the annotation step is manifest in the  
1140 augmented PSVI in a uniform manner. Specifically, let *x* be an  
1141 attribute or element information item which is identified by the  
1142 language-specific processing as containing one or more uses of  
1143 XML namespace prefixes in its [schema normalized value]

1144 property *y*. If any of these uses of XML namespace prefixes in *y* is  
1145 in a form other than a occurrence of a [QName](#), then a fatal error  
1146 occurs. Otherwise, *x* is additionally augmented by the language-  
1147 specific processing with a **[prefix usage locations]** property which  
1148 contains, corresponding to the sequence of all the QNames in *y*,  
1149 an ordered sequence of one or more triples (*offset*, *prefix*,  
1150 *namespace URI*) where

- 1151 1. *offset* is the zero-based offset from the start of *y* of the first  
1152 character of a [QName](#)
- 1153 2. *prefix* is the string value of the [prefix](#) of that QName (not, to  
1154 be clear, including any trailing colon), if any is present, or [no](#)  
1155 [value](#) otherwise.
- 1156 3. *namespace URI* is the in-scope binding of the that XML  
1157 namespace prefix (or the default XML namespace, if *prefix*  
1158 is [no value](#)), or [no value](#) if no such binding exists (which  
1159 necessarily must result from a use of the default XML  
1160 namespace prefix in a context where no declaration for that  
1161 prefix is in scope),

1162 and these triples occur in increasing order by *offset*.

1163 This concludes the specification of the namespace prefix  
1164 desensitization step.

### 1165 **3.4.3 Namespace Attribute Normalization**

1166 The next step in the series of infoset transformations carried out by  
1167 the Schema Centric Canonicalization algorithm is that of  
1168 normalizing the actual XML namespace prefix declarations in use.  
1169 The XML namespace recommendation allows namespaces to be  
1170 multiply declared throughout an XML instance, possibly with  
1171 several and different namespace prefixes used for the same  
1172 namespace. In the canonical representation, we remove this  
1173 flexibility, declaring each namespace just as needed, and using a  
1174 deterministically constructed namespace prefix in such  
1175 declaration. In this procedure, we borrow heavily from some of the  
1176 similar work carried out in the Exclusive XML Canonicalization  
1177 recommendation. We begin with some definitions.

#### 1178 **ancestor information item**

1179 An *ancestor* information item *a* of an information item *i* in an  
1180 infoset is any information item transitively reachable from *i*  
1181 through traversal of the [parent] properties of element,  
1182 processing instruction, unexpanded entity reference,  
1183 character, comment, and document type declaration  
1184 information items, and the [owner element] property of  
1185 attribute information items. Notation, unparsed entity, and

1186 namespace information items have no ancestors, nor do  
1187 attribute information items which appear in elements other  
1188 than in their [attributes] properties. Note that the information  
1189 item *i* is not an ancestor of itself.

#### 1190 **self-ancestor**

1191 A self-ancestor of an information item is either the  
1192 information item itself or an ancestor thereof.

#### 1193 **output parent**

1194 The output parent of an information item *i* in an info set is  
1195 (noting that the ancestor relationship is transitive) the  
1196 *nearest* ancestor of *i* which is an element information item  
1197 whose [omitted] property is false, or **no value** if such an  
1198 ancestor does not exist.

#### 1199 **visibly utilize**

1200 An element information item *e* in an info set is said to *visibly*  
1201 *utilize* an XML namespace prefix *p* if any of the following is  
1202 true:

- 1203 1. the [prefix] property of *e* is identical to *p* (note that  
1204 this includes the case where both are **no value**),
- 1205 2. *e* has a [prefix usage locations] property, and that  
1206 property value contains some triple whose *prefix*  
1207 member is identical to *p*
- 1208 3. there exists an attribute information item *a* in the  
1209 info set whose [owner element] property is *e*, whose  
1210 [omitted] property is false, and either
  - 1211 a. the [prefix] property of *a* is identical to *p*,
  - 1212 b. *a* has a [prefix usage locations] property, and  
1213 that property value contains some triple whose  
1214 *prefix* member is identical to *p*

1215 The execution of the namespace attribute normalization step adds  
1216 **[normalized namespace attributes]** properties to certain element  
1217 information items in the info set. Let *e* be any element information  
1218 item whose [omitted] property is false. Then the [normalized  
1219 namespace attributes] property of *e* is that unordered set of  
1220 attribute information items defined recursively as follows.

1221 Let  $N_e$  be the set of all namespace information items *n* in the [in-  
1222 scope namespaces] property of *e* where *n* is **visibly utilized** by *e*.  
1223 Let  $NA_p$  be the set of attribute information items in the [normalized  
1224 namespace attributes] property of any self-ancestor of *p*, where *p*  
1225 is the **output parent** of *e* and if *p* is not **no value**, or the empty set if  
1226 no such output parent exists. Let  $namespaces(N_e)$  be the set of  
1227 strings consisting of the [namespace name] properties of all  
1228 members of  $N_e$ , and let  $namespaces(NA_p)$  be the set of strings

1229 consisting of the [normalized value] properties of all members of  
1230 *N<sub>A</sub>p*.

1231 For each namespace URI *u* in *namespaces(N<sub>e</sub>)* -  
1232 *namespaces(N<sub>A</sub>p)* (so, the name of each namespace with a prefix  
1233 newly utilized at *e*), the [normalized namespace attributes]  
1234 property of *e* contains an attribute information item whose  
1235 properties are as follows:

- 1236 1. the [namespace name] property is (per XML Infoset)  
1237 "http://www.w3.org/2000/xmlns/"
- 1238 2. the [local name] property is either:
  - 1239 a. a the string "xml" if the namespace value is  
1240 "http://www.w3.org/XML/1998/namespace"
  - 1241 b. a string of the form "n" concatenated the canonical  
1242 lexical representation of a non-negative integer *i* (for  
1243 example "n0", "n1", "n2", and so on) where the  
1244 particular integer *i* in question is chosen as described  
1245 just below.
- 1246 3. the [prefix] property is "xmlns"
- 1247 4. the [normalized value] property is the value *u*.
- 1248 5. the [schema normalized value] property is identical to the  
1249 [normalized value] property
- 1250 6. the remaining properties are as set forth above in the  
1251 specification of conversion of attribute nodes to information  
1252 items.

Deleted:

1253 XML namespace prefixes used in the [normalized namespace  
1254 attributes] property (which are manifest in the [local name]  
1255 properties of the attribute information items contained therein) are  
1256 chosen as follows. Let *e* be any element containing a [normalized  
1257 namespace attributes] property. Let *l* be the ordered list resulting  
1258 from sorting the [normalized namespace attributes] property of *e*  
1259 according to the *sort* function described below. Let *k* be the  
1260 maximum over all the ancestors *a* of *e* of the integers used per (b)  
1261 above to form the [local name] property of any attribute item in the  
1262 [normalized namespace attributes] property of *a*, or -1 if no such  
1263 attribute items exist. Then the attributes of *l*, considered in order,  
1264 use, in order, the integers *k*+1, *k*+2, *k*+3, and so on in the  
1265 generation of their [local name] as per (b) above, excepting only  
1266 that if *wildcardOutputRoot(e)* is true, then (in order to avoid  
1267 collisions) any integer which would result in a [local name] property  
1268 which was the same as the [prefix] property of some namespace  
1269 item in the [in-scope namespaces] property of *e* is skipped.

1270 Now that the declaration of necessary namespace attributes has  
1271 been successfully normalized (and, canonically, the default

1272 namespace has been left undeclared), we apply these  
1273 declarations in the appropriate places by defining appropriate  
1274 **[normalized prefix]** and **[prefix & schema normalized value]**  
1275 properties. Let *info* be any information item in the infoset whose  
1276 [omitted] property is false. Then:

- 1277 1. If *info* is an element or attribute information item whose  
1278 [namespace name] property has **no value**, then the  
1279 **[normalized prefix]** property of *info* exists but is **no value**.
- 1280 2. If *info* is an element or attribute information item whose  
1281 [namespace name] property is not **no value**, then let *a* be  
1282 that namespace declaration attribute in the **[normalized**  
1283 **namespace attributes]** of some self-ancestor of *info* where  
1284 the [normalized value] property of *a* is identical to the  
1285 [namespace name] property of *info* (if no such *a* exists, a  
1286 fatal error occurs. This can occur, for example, if all element  
1287 information items in the infoset are omitted, but some  
1288 attributes are retained.). The **[normalized prefix]** property of  
1289 *info* then exists and is the [local name] property of *a*.

1290 Moreover, if *info* contains a **[prefix usage locations]** property, then  
1291 *info* also contains a **[prefix & schema normalized value]** property  
1292 which is identical to the [schema normalized value] property of *info*  
1293 except for differences formed according to the following procedure.  
1294 Consider in turn each triple *t* found in the [prefix usage locations]  
1295 property of *info*. Let *normalizedPrefixUse(t)* be those characters of  
1296 the [prefix & schema normalized value] property of *info* which  
1297 correspond to the characters of the [schema normalized value]  
1298 property of *info* whose zero-based character-offsets lie in the semi-  
1299 open interval [*offset*, *offset+cch-1+z*), where

- 1300 1. *offset* is the *offset* member of *t*,
- 1301 2. *cch* is the number of characters in the *prefix* member of *t* (if  
1302 *prefix* is not **no value**) or zero (otherwise), and
- 1303 3. *z* is one if *prefix* is not **no value** and the *offset+cch-1+1*'st  
1304 character of the [schema normalized value] of *info* property  
1305 is a colon, and zero otherwise.

1306 Then the characters of *normalizedPrefixUse(t)* are determined as  
1307 follows:

- 1308 1. If the *namespace URI* of *t* has **no value**, then  
1309 *normalizedPrefixUse(t)* is the empty string.
- 1310 2. Otherwise, let *a* be that namespace declaration attribute in  
1311 the [normalized namespace attributes] of some self-  
1312 ancestor of *info* where the [normalized value] property of *a*



1313 is identical to the *namespace URI* of *t* (if no such *a* exists, a  
1314 fatal error occurs). Then *normalizedPrefixUse(t)* is the [local  
1315 name] of *a* followed by a colon.

1316 This completes the specification of the namespace attribute  
1317 normalization step.

### 1318 **3.4.4 Data-type Canonicalization**

1319 The XML Schema Datatypes specification defines for a certain set  
1320 of its built-in data-types a [canonical](#) lexical representation of the  
1321 values of each of those data types. To that identified set of  
1322 canonical representations Schema Centric Canonicalization adds  
1323 several new rules; in some cases, it refines those rules provided  
1324 by XML Schema.

1325 The most complicated part of data type canonicalization lies in  
1326 dealing with character sequences which are as a matter of  
1327 application-level schema design considered to be case insensitive.  
1328 It is important that case-insensitivity of application data be  
1329 integrated into the canonicalization algorithm: if it were not, then  
1330 applications may be forced to remember the exact case used for  
1331 certain data when they otherwise would not need to, a requirement  
1332 which may well be in tension with the application semantic of case-  
1333 insensitivity, and thus quite possibly a significant implementation  
1334 burden.

1335 The relevant technical reference for case-mapping considerations  
1336 for Unicode characters is a [technical report](#) published by the  
1337 Unicode Consortium. Case-mapping of Unicode characters is  
1338 more subtle than readers might naively intuit from their personal  
1339 experience. The mapping process [can at times](#) be both locale-  
1340 specific (Turkish has special considerations, for example) and  
1341 context-dependent (some characters case-map differently  
1342 according to whether they lie at the end of a word or not). Mapping  
1343 of case can change the length of a character sequence. Upper and  
1344 lower cases are not precise duals: there exist pairs of strings which  
1345 are equivalent in their upper case-mapping but not in their lower  
1346 case, and visa versa.

1347 In order to accommodate these flexibilities, we define several  
1348 attributes within the Schema Centric Canonicalization algorithm  
1349 [namespace](#) in order to assist with the identification of data which is  
1350 to be considered case-insensitive and the precise manner in which  
1351 that is to be carried out. As was the case for the [embeddedLang](#)  
1352 and [embeddedLangAttribute](#) attributes previously defined, these  
1353 attributes are intended to be used as annotations of relevant  
1354 schema components.

1355 The **caseMap** attribute, which is of type  
1356 [language](#), is defined in the Schema  
1357 Centric Canonicalization algorithm  
1358 namespace. When used as an attribute  
1359 annotation to a schema component, a  
1360 caseMap attribute indicates that case-  
1361 mapping is to be performed on data  
1362 which validates against the schema  
1363 component according to the case-  
1364 mapping rules of the fixed locale  
1365 identified by the value of the attribute.

1366 The **caseMapAttribute** attribute, which  
1367 is of type [QName](#), is defined in the  
1368 Schema Centric Canonicalization  
1369 algorithm namespace. When used as  
1370 an attribute annotation to a schema  
1371 component, a caseMapAttribute  
1372 attribute indicates that an information  
1373 item which validates against the  
1374 schema component in question is to be  
1375 case mapped during the  
1376 canonicalization process according to  
1377 the rules of the locale which is  
1378 dynamically indicated in the information  
1379 item (necessarily an element  
1380 information item) as the value of a  
1381 certain attribute thereof, namely the  
1382 attribute whose qualified name is  
1383 indicated in the value of the  
1384 caseMapAttribute attribute (which must  
1385 be of type [language](#) or a restriction  
1386 thereof).

1387 The **caseMapKind** attribute, which is of  
1388 type [string](#) but restricted to the  
1389 enumerated values "upper", "lower",  
1390 and "fold", is defined in the Schema  
1391 Centric Canonicalization algorithm  
1392 namespace. When used as an attribute  
1393 annotation to a schema component, a  
1394 caseMapKind attribute indicates  
1395 whether upper-case or lower-case  
1396 mapping or case-folding is to be carried  
1397 out as part of the canonicalization  
1398 process. If this attribute is contextually  
1399 absent but at least one of caseMap or

1400 caseMapAttribute is contextually  
1401 present, upper-case mapping is carried  
1402 out.

1403 Traditional ASCII-like case insensitivity can be most easily  
1404 approximated by simply specifying "fold" for the caseMapKind  
1405 attribute and omitting both caseMap and caseMapAttribute. Also,  
1406 schema designers are cautioned to be careful in combining case-  
1407 mapping annotations together with length-limiting facets of strings  
1408 and URIs, due to the length-adjustment that may occur during  
1409 canonicalization.

1410 The data-type canonicalization step of Schema Centric  
1411 Canonicalization is carried out according to the following rules:

- 1412 1. Per the relevant [clarification](#) E2-9 in the [errata](#) to XML  
1413 Schema, the canonical lexical representation of a datum of  
1414 type [base64Binary](#) must conform to the grammatical  
1415 production *Canonical-base64Binary* as defined therein.  
1416 That production permits in the representation only valid  
1417 base64 encodings which *only* contain characters from the  
1418 base64 alphabet as defined by section "6.8 Base64  
1419 Content-Transfer-Encoding" of [RFC 2045](#) (in particular,  
1420 whitespace characters are not in the alphabet), excepting  
1421 only that the representation is to be formed into lines of  
1422 exactly 76 characters (except for the last line, which must  
1423 be 76 characters or shorter) by the appropriate periodic  
1424 occurrence of a line-feed character (that is, the character  
1425 whose character code is (decimal) 10) at the end of each  
1426 line (including the last).
- 1427 2. The canonical lexical representation of a datum of type  
1428 [dateTime](#) permits only the lexical representation 00:00:00 to  
1429 denote a time value of midnight (that is, the representation  
1430 24:00:00 is prohibited). Further (per XML Schema) either  
1431 the time zone must be omitted or, if present, the time zone  
1432 must be Coordinated Universal Time (UTC) indicated by a  
1433 "Z".
- 1434 3. The canonical lexical representation of a datum of type [float](#)  
1435 or [double](#) is defined by prohibiting certain options from the  
1436 [lexical representation](#). Specifically, the exponent must be  
1437 indicated by "E". Leading zeroes and the preceding optional  
1438 "+" sign are prohibited in the exponent. For the mantissa,  
1439 the preceding optional "+" sign is prohibited and the decimal  
1440 point is required. For the exponent, the preceding optional  
1441 "+" sign is prohibited. Leading and trailing zeroes are  
1442 prohibited subject to the following: number representations  
1443 must be normalized such that there is a single digit to the

1444 left of the decimal point and at least a single digit to the right  
1445 of the decimal point such that the number of of leading  
1446 zeros in the overall sequence of such digits is a small as  
1447 otherwise possible.

- 1448 4. The canonical lexical representation of a datum of type  
1449 [language](#) permits only the use of upper case characters.
- 1450 5. The canonical lexical representation of a datum of type  
1451 [gYearMonth](#) and [gYear](#) prohibits the use of leading zeros  
1452 for values where the absolute value of the year in question  
1453 is outside the range of 0001 to 9999.
- 1454 6. The canonical lexical representation of an element or  
1455 attribute information item *info* which of type [string](#) or [anyUri](#)  
1456 or a restriction thereof and where either of the following is  
1457 true:
  - 1458 a. the following is true
    - 1459 i. [getAnnot](#)(*info*, "caseMap", [sccns](#)) is present, or, if  
1460 not
    - 1461 ii. [getAnnot](#)(*info*, "caseMapAttribute", [sccns](#)) is  
1462 present
  - 1463 b. [getAnnot](#)(*info*, "caseMapKind", [sccns](#)) is present

1464 is the result of the application of the function [caseMap](#) with  
1465 the parameters

- 1466 c. the sequence of characters comprising the value of  
1467 the element or attribute in question,
- 1468 d. the language indicated according to the applicable  
1469 case i. or ii. above, if any, or the value [absent](#)  
1470 otherwise,
- 1471 e. [getAnnot](#)(*info*, "caseMapKind", [sccns](#)).

- 1472 7. If none of the preceding rules apply, the canonical lexical  
1473 representation of a datum of [primitive type](#) for which XML  
1474 Schema Datatypes defines a canonical lexical  
1475 representation is the representation defined therein.
- 1476 8. If none of the preceding rules apply, the canonical lexical  
1477 representation of a datum which is of a [primitive type](#) is the  
1478 not-further-processed representation of the datum itself.
- 1479 9. The canonical lexical representation of a datum of a type  
1480 which is derived by [list](#) is that which is defined by the XML  
1481 Schema Datatypes specification (note that this includes the  
1482 collapsing of the whitespace therein).
- 1483 10. If none of the preceding rules apply, the canonical lexical  
1484 representation of a datum which is of a simple type that is a  
1485 restriction of a type for which a canonical lexical  
1486 representation is defined is the representation of the datum

1487 according to the canonical lexical representation so defined  
1488 for that base type.

1489 Thus, a canonical lexical representation for all non-union simple  
1490 types is defined.

1491 The function *caseMap* takes three input parameters:

- 1492 1. a sequence of characters whose case is to be mapped,
- 1493 2. a locale in the form of a [language](#) in whose context the  
1494 mapping is to be carried out, or the value [absent](#), which is  
1495 to be treated as if "en" were provided,
- 1496 3. either the string "upper", the string "lower", the string "fold",  
1497 or the value [absent](#), indicating whether upper-case or lower-  
1498 case mapping or [case-folding](#) is to be carried out; the value  
1499 [absent](#) is treated as if "upper" were provided.

1500 The upper-case or lower-case mapping process of the *caseMap*  
1501 function is carried out in the context of the indicated locale  
1502 according to the (respectively) `UCD_upper()` or `UCD_lower()`  
1503 functions as [specified](#) by the Unicode Consortium. The case-  
1504 folding process is carried out by mapping characters through the  
1505 [CaseFolding.txt](#) file in the Unicode Character Database as  
1506 [specified](#) by the Unicode Consortium.

1507 To carry out the data-type canonicalization step in the Schema  
1508 Centric Canonicalization algorithm, the [schema normalized value]  
1509 property of all [element](#) and [attribute](#) information items in the output  
1510 of the namespace attribute normalization step whose [member  
1511 type definition] (if present) or [type definition] (otherwise) property  
1512 is a simple type is replaced by the defined canonical lexical  
1513 representation of the member of the relevant [value space](#) which is  
1514 represented by the [schema normalized value].

1515 The infoSet which is output from the data-type canonicalization  
1516 step is the schema-canonicalized infoSet.

### 1517 **3.5 Serialization of the Schema-** 1518 **Canonicalized InfoSet**

1519 The final step in the Schema Centric Canonicalization algorithm is  
1520 the serialization of the schema-canonicalized infoSet into a  
1521 sequence of octets.

1522 In the description of the serialization algorithm that follows, at  
1523 various times a statement is made to the effect that a certain  
1524 sequence of characters is to be emitted or output. In all cases, it is

1525 to be understood that the actual octet sequences emitted are the  
1526 corresponding UTF-8 representations of the characters in  
1527 question. The character referred to as "space" has a character  
1528 code of (decimal) 32, the character referred to as "colon" has a  
1529 character code of (decimal) 58, and the character referred to as  
1530 "quote" has a character code of (decimal) 34.

1531 Also, the algorithm description makes use of the notation  
1532 "*info[propertyName]*". This is to be understood to represent the  
1533 value of the property whose name is *propertyName* on the  
1534 information item *info*.

1535 The serialization of the schema-canonicalized infoset, and thus the  
1536 output of the overall Schema Centric Canonicalization algorithm, is  
1537 defined to be the octet sequence that results from the function  
1538 invocation *serialize(d)*, where *d* is the document information item in  
1539 the schema-canonicalized infoset, and *serialize* is the function  
1540 defined below.

### 1541 3.5.1 The function *serialize*

1542 The *serialize* function is defined recursively in terms of the  
1543 serialization of individual types of information item. Let the  
1544 functions *recurse*, *sort*, *escape*, *wildcarded*, and  
1545 *wildcardOutputRoot* be defined as set forth later. Let *info* be an  
1546 arbitrary information item. Let *serialize* be the function taking an  
1547 information item as input and returning an sequence of octets as  
1548 output which is defined as follows.

- 1549 1. If *info* is a **document information item**, then *serialize(info)*  
1550 is the in-order concatenation of the following:
  - 1551 a. if *info[omitted]* is false, and if either *info[notations]* or  
1552 *info[unparsed entities]* contains a notation or an  
1553 unparsed entity information item (respectively) whose  
1554 *[omitted]* property is false, then
    - 1555 i. the characters "<!DOCTYPE "
    - 1556 ii. the appropriate case from the following
      - 1557 1. if *wildcarded(info[document element])* is  
1558 false, then if *info[document element][normalized prefix]* is not **no**  
1559 **value**, then the characters thereof,  
1560 followed by a colon
      - 1561 2. if *wildcarded(info[document element])* is  
1562 true, then if *info[document element][prefix]* is not **no value**, then  
1563 the characters thereof, followed by a  
1564 colon



1613 where c[omitted] is true, then the  
1614 empty octet sequence,  
1615 b. otherwise, escape(info[[prefix &](#)  
1616 [schema normalized value](#)])  
1617 2. else if the property info[[schema](#)  
1618 [normalized value](#)] is present, then  
1619 a. if info[children] contains any  
1620 character information item c  
1621 where c[omitted] is true, then the  
1622 empty octet sequence,  
1623 b. otherwise, escape(info[[schema](#)  
1624 [normalized value](#)]),  
1625 3. else if at least one member of  
1626 info[children] is an element information  
1627 item which possesses a [[validating](#)  
1628 [model group all](#)] property, then let the  
1629 subsequence of info[children]  
1630 consisting of all those elements which  
1631 possess a [[validating model group all](#)]  
1632 property be partitioned into into  $k$   
1633 subsequences  $l_1$  to  $l_k$  such that  $k$  is as  
1634 small as possible and all items of a  
1635 given subsequence share the same  
1636 model group information item for their  
1637 [[validating model group all](#)] property  
1638 (XML Schema assures that this is well-  
1639 defined), and let *children'* be a re-  
1640 ordering of info[children] according to  
1641 the following constraints:  
1642 a. if an item  $c$  of info[children]  
1643 possesses a [[validating model](#)  
1644 [group all](#)] property, and is  
1645 therefore contained in  
1646 subsequence  $l_i$  for some  $i$ , then  
1647 the relative order of  $c$  in *children'*  
1648 with respect to  
1649 i. any item  $d$  of  $l_i$  different than  
1650  $c$  is the same as the  
1651 relative ordering of  $c$  and  
1652  $d$  in  $sort(l_i)$   
1653 ii. any item  $e$  of  $l_j$  (for some  $i \neq j$ )  
1654 is the same as the relative  
1655 ordering of the first items  
1656 of  $l_i$  and  $l_j$   
1657 iii. any other item  $f$  of  
1658 info[children] is the same



1659 as the relative ordering in  
1660 info[children] of *f* with that  
1661 item *g* of *l<sub>i</sub>* where the  
1662 index of *g* in *l<sub>i</sub>* is the same  
1663 as the index of *c* in *sort(l<sub>i</sub>)*  
1664 b. if items *m* and *n* of info[children]  
1665 do not possess a [validating  
1666 model group all] property, then  
1667 they occur in *children'* in the  
1668 same relative order as they  
1669 occur as items in info[children]

1670 then, recurse(children')

1671 4. otherwise, if info[content type] is element-only,  
1672 then recurse(nwsChildren), where nwsChildren  
1673 is the result of removing from info[children]  
1674 those character information items whose  
1675 [character code] is defined as a white space in  
1676 the XML 1.0 Recommendation (this reflects  
1677 the validation rule in clause 2.3 of §3.4.4 of  
1678 XML Schema).

1679 5. otherwise, recurse(info[children])

1680 v. if info[omitted] is false, then

- 1681 1. the characters "</"
- 1682 2. the appropriate case from the following:
  - 1683 a. if *wildcarded(info)* is false, then if
  - 1684 info[normalized prefix] is not **no**
  - 1685 **value**, then the characters
  - 1686 thereof, followed by a colon
  - 1687 b. if *wildcarded(info)* is true, then if
  - 1688 info[prefix] is not **no value**, then
  - 1689 the characters thereof, followed
  - 1690 by a colon
- 1691 3. the characters of info[local name]
- 1692 4. the character ">"

1693 3. If *info* is an **attribute information item**, then *serialize(info)*  
1694 is the in-order concatenation of the following:

- 1695 a. if info[omitted] is false, then
  - 1696 i. the character space
  - 1697 ii. the appropriate case from the following:
    - 1698 1. if *wildcarded(info)* is false, then if
    - 1699 info[normalized prefix] is not **no value**,
    - 1700 then the characters thereof, followed by
    - 1701 a colon

1702                                   2. if *wildcarded(info)* is true, then if  
1703                                    info[*prefix*] is not **no value**, then the  
1704                                    characters thereof, followed by a colon  
1705                           iii. the characters of info[*local name*]  
1706                           iv. the character "="  
1707                           v. the character quote  
1708                           vi. the appropriate case of the following:  
1709                                1. if the property info[**prefix & schema**  
1710                                **normalized value**] is present, then  
1711                                escape(info[**prefix & schema**  
1712                                **normalized value**])  
1713                                2. if info[*schema normalized value*] exists,  
1714                                then escape(info[**schema normalized**  
1715                                **value**])  
1716                                3. otherwise (the attribute was  
1717                                *wildcarded*), escape(info[*normalized*  
1718                                *value*])  
1719                                vii. the character quote  
1720                           b. otherwise, the empty octet sequence

1721   4. If *info* is a **namespace information item**, then  
1722    *serialize(info)* is the in-order concatenation of the following:  
1723    a. if info[*omitted*] is false, then  
1724        i. the character space  
1725        ii. the characters "xmlns:"  
1726        iii. the characters of info[*prefix*]  
1727        iv. the character "="  
1728        v. the character quote  
1729        vi. escape(info[*namespace name*])  
1730        vii. the character quote  
1731    b. otherwise, the empty octet sequence

1732   5. If *info* is an **unparsed entity information item**, then  
1733    *serialize(info)* is the in-order concatenation of the **following**:  
1734    a. if info[*omitted*] is false, then  
1735        i. the characters "<!ENTITY"  
1736        ii. the character space  
1737        iii. info[*name*]  
1738        iv. the character space  
1739        v. the appropriate case of the following  
1740            1. if info[*public identifier*] is not **no value**,  
1741            then the in-order concatenation of the  
1742            following:  
1743                a. "PUBLIC"  
1744                b. the character space  
1745                c. info[*public identifier*]  
1746                d. the character space

1747 e. info[system identifier]  
1748 2. otherwise, the in order concatenation of  
1749 the following:  
1750 a. "SYSTEM"  
1751 b. the character space  
1752 c. info[system identifier]  
1753 vi. if info[notation name] is not **no value**, then the in-  
1754 order concatenation of the following:  
1755 1. the character space  
1756 2. "NDATA"  
1757 3. the character space  
1758 4. info[notation name]  
1759 vii. the character ">"  
1760 b. otherwise, the empty octet sequence

1761 6. If *info* is a **notation information item**, then *serialize(info)* is  
1762 the in-order concatenation of the following:  
1763 a. if info[omitted] is false, then  
1764 i. the characters "<!NOTATION"  
1765 ii. the character space  
1766 iii. info[name]  
1767 iv. the character space  
1768 v. the appropriate case of the following  
1769 1. if info[public identifier] and info[system  
1770 identifier] are not both **no value**, then  
1771 the in-order concatenation of the  
1772 following:  
1773 a. "PUBLIC"  
1774 b. the character space  
1775 c. info[public identifier]  
1776 d. the character space  
1777 e. info[system identifier]  
1778 2. else if info[public identifier] has **no**  
1779 **value**, the in-order concatenation of the  
1780 following:  
1781 a. "SYSTEM"  
1782 b. the character space  
1783 c. info[system identifier]  
1784 3. otherwise, the in-order concatenation of  
1785 the following  
1786 a. "PUBLIC"  
1787 b. the character space  
1788 c. info[public identifier]  
1789 vi. the character ">"  
1790 b. otherwise, the empty octet sequence

1791 7. Otherwise (this includes processing instruction,  
1792 unexpanded entity reference, character, comment, and  
1793 document type declaration information items, though  
1794 characters and DTD's are accounted for by other means),  
1795 *serialize(info)* is the empty sequence of octets.

### 1796 **3.5.2 The function recurse**

1797 The function *recurse* is a function which takes as input an ordered  
1798 list *infos* of information items and proceeds as follows.

1799 First, character information items in *infos* whose [omitted] property  
1800 is 'true' are pruned by removing them from the list. Next, the  
1801 pruned list is divided into an ordered sequence of sub-lists  $l_1$   
1802 through  $l_k$  according to the rule that a sub-list which contains  
1803 character items may not contain other types of information items,  
1804 but otherwise  $k$  is as small as possible. The result of *recurse* is  
1805 then the in-order concatenation of processing in order each sub-  
1806 list  $l_i$  in turn in the following manner:

- 1807 1. If  $l_i$  contains character information items, then let  $s_i$  be the  
1808 string of characters of length equal to the size of  $l_i$  where the  
1809 ISO 10646 character code of the  $n$ th character of  $s_i$  is equal  
1810 to the [character code] property of the  $n$ th character of  $l_i$ .  
1811 The output of processing  $l_i$  is then the result of the function  
1812 invocation *escape*( $s_i$ ).
- 1813 2. If  $l_i$  does not contain character information items, then the  
1814 output of processing  $l_i$  is the in-order concatenation of  
1815 *serialize(info)* as *info* ranges in order over the information  
1816 items in the sub-list  $l_i$ .

### 1817 **3.5.3 The function escape**

1818 The function *escape* is that function which takes as input a string  $s$   
1819 and returns a copy of  $s$  where each occurrence of any of the five  
1820 characters &lt; > ' " in  $s$  is replaced by its corresponding predefined  
1821 entity.

### 1822 **3.5.4 The functions sort and compare**

1823 The function *sort* takes as input an unordered set or an ordered list  
1824 of information items and returns an ordered list of those  
1825 information items arranged in increasing order according to the  
1826 function *compare*, unless some of the information items do not  
1827 have a relative ordering, in which case a fatal error occurs.

1828 The function *compare* takes two information items  $a$  and  $b$  as input  
1829 and returns an element of {*less than or equal*, *greater than or*  
1830 *equal*, *no relative ordering*} as output according to the following:

- 1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852
1. If  $a$  and  $b$  are both **attribute information items**, then (as in Canonical XML) *less than or equal* or *greater than or equal* is returned according to a lexicographical comparison with the [namespace name] property as the primary key and the [local name] as the secondary key.
  2. If  $a$  and  $b$  are both **element information items**, then *less than or equal* or *greater than or equal* is returned according to a lexicographical comparison with the [namespace name] property as the primary key and the [local name] as the secondary key.
  3. If  $a$  and  $b$  are both **namespace information items**, then *less than or equal* or *greater than or equal* is returned according to a lexicographical comparison with the [namespace name] property as the primary key and the [prefix] property as the secondary key.
  4. If  $a$  and  $b$  are both **notation information items**, then *less than or equal* or *greater than or equal* is returned according to a comparison of their [name] properties
  5. If  $a$  and  $b$  are both **unparsed entity information items**, then *less than or equal* or *greater than or equal* is returned according to a comparison of their [name] properties
  6. Otherwise, *no relative ordering* is returned.

### 1853 **3.5.5 The function wildcarded**

1854 The function *wildcard* takes an element or an attribute information  
1855 as input and returns a boolean indicating whether validation was  
1856 not attempted on that item. In the Schema Centric  
1857 Canonicalization algorithm, validation of an information item will  
1858 only not be attempted as a consequence of the item or a parent  
1859 thereof being validated against a **wildcard** whose {process  
1860 contents} property is either *skip* or *lax*.

1861 Let  $i$  be the information item input to *wildcarded*. The function is  
1862 then defined as follows:

- 1863  
1864
1. If  $\neg$ [validation attempted] is *none*, then *true* is returned.
  2. Otherwise, *false* is returned.

### 1865 **3.5.6 The function wildcardOutputRoot**

1866 The function *wildcardOutputRoot* takes an element item as input  
1867 and returns a boolean indicating whether the item is an  
1868 appropriate one on which to place the contextual namespace  
1869 declarations necessary for dealing with wildcarded items contained  
1870 therein. Let  $e$  be the information item input to *wildcardOutputRoot*.  
1871 The function is then defined as follows:

- 1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880
1. If *e*[omitted] is true, then *false* is returned.
  2. If *wildcarded(e)* is *false* and *e*[attributes] contains any attribute items *a* for which *wildcarded(a)* is *true*, then *true* is returned.
  3. If *wildcarded(e)* is *true*, and there does not transitively exist any [parent] element item *p* of *e* where either the preceding clause (2) applies or both *p*[omitted] is *false* and *wildcarded(p)* is *true*, then *true* is returned.
  4. Otherwise, *false* is returned.

1881

## 4. Use of Schema Centric Canonicalization in XML Security

1882

### 4.1 Algorithm Identification

1883

1884

1885

1886

1887

1888

The XML-Signature Syntax and Processing [recommendation](#) (XML DSIG) defines the notion of a [canonicalization algorithm](#) together with the use of URIs as identifiers for such algorithms. In XML DSIG, the use of canonicalization algorithms is architected in three places:

1889

1890

1891

1892

1893

1894

1895

1896

1897

1898

1899

1900

1901

1. As part of the signature [generation](#) and [validation](#) processes, where it is used to canonicalize a [SignedInfo](#) element prior to its being fed into a digest algorithm.
2. As a [Transform algorithm](#) in the pipeline of Transforms inside a [Reference](#), used to modify data during the reference [generation](#) and [validation](#) processes. As a matter of good XML DSIG hygiene, such a canonicalization Transform should always be used in the pipeline, and in fact should always occur as the last Transform therein.
3. As the [means](#) by which a Transform in the pipeline which requires an octet stream as input but is instead presented (by the previous Transform) with an input node-set converts the latter into the former.

1902

[XML Encryption](#) makes similar use of these algorithms.

1903

1904

1905

1906

1907

This specification asserts that the [URI of the Schema Centric Canonicalization algorithm namespace](#) is the identifier (in the sense of XML DSIG) of a canonicalization algorithm. This identifier denotes the Schema Centric Canonicalization algorithm. The algorithm does not require or permit any explicit parameters.

1908

### 4.2 Re-Enveloping of Canonicalized Data

1909

1910

As is [discussed](#) in Exclusive XML Canonicalization, many applications from time to time find it useful to be able to change

1911 the enveloping context of a subset of an XML document without  
1912 changing the canonical form thereof.

1913 In such situations, if Schema Centric Canonicalization is the  
1914 algorithm of relevance, then applications SHOULD avoid  
1915 references to notations or unparsed entities in the document  
1916 subset in question, since the canonical representation of the  
1917 notation and entity declarations referred to (which must, for  
1918 security, be part of the canonical form) are defined in a document  
1919 type declaration, the presence of which significantly complicates  
1920 the task of re-enveloping.

## 1921 5. Resolutions

1922 This section discusses a few key decision points as well as a  
1923 rationale for each decision.

### 1924 5.1 No Non-Schema-Influencing Information 1925 Items

1926 Several of the eleven different types of information items either  
1927 can never appear in an infoset which successfully validates  
1928 according to XML Schema or can in no way affect the outcome  
1929 thereof. Accordingly, representations of such information items  
1930 never appear in the output of the Schema Centric Canonicalization  
1931 algorithm. These types of information item are the following:

- 1932 1. **comment** information items and **processing instruction**  
1933 information items: as is [described](#) in the XML Schema  
1934 Structures recommendation, comments and processing  
1935 instructions, even in the midst of text, are ignored for all  
1936 validation purposes. Thus, for example, each can appear in  
1937 such places as the middle of the sequence of digits of an  
1938 integer which is the content of an element with an integral  
1939 simple type. Were it required (or even optional) to preserve  
1940 the significance of such items with respect to the  
1941 canonicalization, applications, particularly those wishing to  
1942 [shred](#) XML information into a relational or other store, would  
1943 face cumbersome and significant impediments to  
1944 implementation.
- 1945 2. **unexpanded entity reference** information items: as is  
1946 [explained](#) in the XML Infoset recommendation, a validating  
1947 XML processor will never generate unexpanded entity  
1948 reference information items for a valid document.
- 1949 3. **document type declaration** information items: these are  
1950 excluded since all possible effects of their processing are  
1951 modeled in various properties of other information items.

1952  
1953  
1954  
1955

## 5.2 No Special Whitespace Processing

Believing their reasoning to be sound, we adopt the attitude of Canonical XML towards the processing of whitespace in character content, namely that no special processing is carried out:

1956  
1957  
1958  
1959  
1960  
1961

"All whitespace within the root document element MUST be preserved (except for any #xD characters deleted by line delimiter normalization). This includes all whitespace in external entities."

1962  
1963  
1964

Moreover, for analogous reasons, we adopt the attitude of Exclusive XML Canonicalization towards the lack of special processing of the [xml:lang](#) and the [xml:space](#) attributes.

1965  
1966  
1967  
1968

It is perhaps worth noting by way of contrast that (unrelated to [xml:lang](#) and [xml:space](#)) XML Schema [defines](#) certain whitespace processing rules of its own; these are, of course, carried out by Schema Centric Canonicalization.

1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979

## 5.3 Case-Mapping vs. Case-Folding

The [Unicode Technical Report](#) on Case Mappings distinguishes case-mapping from a similar process termed [case-folding](#). Unlike case-mapping, case-folding is a locale-independent operation, and does not encounter the issue that strings may be equal or differ depending on the direction in which they are case-mapped. As is clear in the report, case-folding suffers from being only an approximation to language-specific rules of processing, and is primarily aimed at legacy systems where locale information simply is not feasibly available with which to do a more complete processing.

1980  
1981

The Schema Centric Canonicalization algorithm supports the use of either case-mapping or case-folding in user schemas.

1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991

## 5.4 No Canonicalization of anyURI Datatype

XML Schema Datatypes does not define a canonical lexical representation for data of type [anyURI](#). In the present specification, thought was given to reconsidering this position. As is described in the [specification](#) of Uniform Resource Identifiers, various aspects of the syntactic structure of URIs are considered case insensitive: the scheme part of the URI is an example (or *probably* is one: contrast §3.1 with §6 in RFC2396 with respect to this point), and various particular schemes have substructure that is so. Moreover, a subset of URI share a common syntax for



1992 representing hierarchical relationships within their namespace, and  
1993 for the relative (as opposed to absolute) form of such URI, an  
1994 algorithm exists (see §5.2 of RFC2396) by which certain aspects  
1995 of the URI representation involving "." and ".." are canonicalized.

1996 For these and related reasons it is reasonable to ask whether a  
1997 canonical lexical representation for data of type anyURI should be  
1998 specified. This, however, is a difficult if not insurmountable task.  
1999 Many of the details of an appropriate canonicalization (such as  
2000 case-mapping or case-folding) are inherently scheme-specific, and  
2001 it is intrinsically impossible for any one Schema Centric  
2002 Canonicalization implementation to understand the universe of  
2003 possible URI schemes it might encounter (and so canonicalize  
2004 them all appropriately). Even for some commonly known URI  
2005 schemes, the relevant specifications lack crisp clarity on some  
2006 germane issues. And the algorithm of §5.2 of RFC2396 can (see  
2007 *ibid*, §5.1) only be carried out in the context of a specific base URI;  
2008 as generally speaking such relevant base URI may be application-  
2009 level notions not represented in XML, the algorithm of §5.2 must  
2010 remain out of scope so far as XML canonicalization is concerned.

2011 These reasons, together with the lack of compelling pragmatic  
2012 problems caused by simply having all [anyURI](#) data canonicalize to  
2013 itself, indicate that the prudent course of action is that Schema  
2014 Centric Canonicalization should not differ from XML Schema  
2015 Datatypes on this issue.

## 2016 6. References

### 2017 Keywords

2018 [RFC 2119](#). *Key words for use in RFCs to Indicate*  
2019 *Requirement Levels*. Best Current Practice. S. Bradner.  
2020 March 1997. S. Bradner. March 1997.  
2021 <http://www.ietf.org/rfc/rfc2119.txt>

### 2022 Unicode

2023 *Unicode 3.1*. The Unicode Consortium.  
2024 <http://www.unicode.org/unicode/reports/tr27/>.

### 2025 Unicode Normalization

2026 *Unicode Normalization Forms*. The Unicode Consortium.  
2027 <http://www.unicode.org/unicode/reports/tr15/>.

### 2028 Unicode Case Mappings

2029 *Case Mappings*. The Unicode Consortium.  
2030 <http://www.unicode.org/unicode/reports/tr21/>.

### 2031 URI

2032 [RFC 2396](#). *Uniform Resource Identifiers (URI): Generic*  
2033 *Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. August

2034 1998. See also [RFC 2732](#). *Format for Literal IPv6*  
2035 *Addresses in URL's*. R. Hinden et al.  
2036 <http://www.ietf.org/rfc/rfc2396.txt>. See also  
2037 <http://www.ietf.org/rfc/rfc2732.txt>.  
2038 **XML**  
2039 [Extensible Markup Language \(XML\) 1.0 \(Second Edition\)](#).  
2040 W3C Recommendation. T. Bray, E. Maler, J. Paoli, C. M.  
2041 Sperberg-McQueen. October 2000.  
2042 <http://www.w3.org/TR/2000/REC-xml-20001006>.  
2043 **XML-C14N**  
2044 [Canonical XML](#). W3C Recommendation. J. Boyer. March  
2045 2001.  
2046 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>  
2047 <http://www.ietf.org/rfc/rfc3076.txt>  
2048 **XML-DSig**  
2049 [XML-Signature Syntax and Processing](#). W3C  
2050 Recommendation. D. Eastlake, J. Reagle, and D. Solo.  
2051 12 February 2002.  
2052 <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>  
2053 **XML-Enc**  
2054 [XML Encryption Syntax and Processing](#). D. Eastlake, and J.  
2055 Reagle. W3C Candidate Recommendation. 04 March 2002.  
2056 <http://www.w3.org/TR/2002/CR-xmlenc-core-20020304>  
2057 **XML-Exc-C14N**  
2058 [Exclusive XML Canonicalization](#) W3C Candidate  
2059 Recommendation. J. Boyer, D. Eastlake, and J. Reagle.  
2060 12 February 2002.  
2061 <http://www.w3.org/TR/2002/CR-xml-exc-c14n-20020212>  
2062 **XML-Infoset**  
2063 *XML Information Set*, John Cowan and Richard Tobin, eds.,  
2064 W3C, 24 October 2001. See  
2065 <http://www.w3.org/TR/2001/REC-xml-infoset-20011024/>  
2066 **XML-NS**  
2067 [Namespaces in XML](#). Recommendation. T. Bray, D.  
2068 Hollander, and A. Layman. January 1999.  
2069 <http://www.w3.org/TR/1999/REC-xml-names-19990114/>  
2070 **XML-Schema**  
2071 [XML Schema](#). Recommendation. H. Thompson, D. Beech,  
2072 M. Maloney, N. Mendelsohn. 2 May 2001.  
2073 <http://www.w3.org/XML/Schema>  
2074 **XML-Schema-Errata**  
2075 [XML Schema 1.0 Specification Errata](#).  
2076 <http://www.w3.org/2001/05/xmlschema-errata>  
2077 **XPath**

2078 XML Path Language (XPath) Version 1.0 , W3C  
2079 Recommendation. eds. James Clark and Steven DeRose.  
2080 16 November 1999.  
2081 <http://www.w3.org/TR/1999/REC-xpath-19991116>.

2082

## 7. Revision History

- 13 February 2002 Initial distribution for public review.
- 15 May 2002 Update references to XML-DSIG, XML-Exc-C14N, XML-Enc to refer to updated publications. Revise commentary thereon in introduction to reflect changes in these updates.  
Allow embeddedLang attribute to be used on schema instances as well as schemas themselves. Added by-fiat identification of uses of XPath in XML Schema itself.  
Clarified position with respect to (non)canonicalization of anyURI.  
Expanded limitations section per observations of Exclusive XML Canonicalization and others.  
QNames (and derivations and lists thereof) were not being namespace prefix desensitized. Fixed.  
Added pointer to supporting .xsd file.
- 20 May 2003 Reformatted per OASIS requirements. A very few minor editorial fixes, updated legal language.

2083

## Appendix A: Notices

2084 *Copyright © 2000-2002 by Accenture, Ariba, Inc., Commerce One,*  
2085 *Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies,*  
2086 *Inc., Intel Corporation, International Business Machines*  
2087 *Corporation, Oracle Corporation, SAP AG, Sun Microsystems,*  
2088 *Inc., VeriSign, Inc., and / or Microsoft Corporation. All Rights*  
2089 *Reserved.*

2090 This document is provided by the companies named above  
2091 ("Licensors") under the following license. By using and/or copying  
2092 this document, or the document from which this statement is  
2093 linked, you (the licensee) agree that you have read, understood,  
2094 and will comply with the following terms and conditions:  
2095 Permission to use, copy, and distribute the contents of this  
2096 document, or the document from which this statement is linked, in  
2097 any medium for any purpose and without fee or royalty under  
2098 copyrights is hereby granted, provided that you include the  
2099 following on ALL copies of the document, or portions thereof, that  
2100 you use:

- 2101 1. A link to the original document.  
2102 2. An attribution statement: "*Copyright © 2000-2002 by*  
2103 *Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited,*  
2104 *Hewlett-Packard Company, i2 Technologies, Inc., Intel*  
2105 *Corporation, International Business Machines Corporation,*  
2106 *Oracle Corporation, SAP AG, Sun Microsystems, Inc.,*  
2107 *VeriSign, Inc., and / or Microsoft Corporation. All Rights*  
2108 *Reserved.*" If the Licensors own any patents or patent  
2109 applications which that may be required for implementing  
2110 and using the specifications contained in the document in  
2111 products that comply with the specifications, upon written  
2112 request, a non-exclusive license under such patents shall  
2113 be granted on reasonable and non-discriminatory terms.

2114 THIS DOCUMENT IS PROVIDED "AS IS," AND LICENSORS  
2115 MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS  
2116 OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,  
2117 WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  
2118 PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;  
2119 THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE  
2120 FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF  
2121 SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY  
2122 PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.  
2123 LICENSORS WILL NOT BE LIABLE FOR ANY DIRECT,  
2124 INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES  
2125 ARISING OUT OF ANY USE OF THE DOCUMENT OR THE  
2126 PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS  
2127 THEREOF.

2128 *Copyright © OASIS Open 2002-2003. All Rights Reserved. OASIS*  
2129 *takes no position regarding the validity or scope of any intellectual*  
2130 *property or other rights that might be claimed to pertain to the*  
2131 *implementation or use of the technology described in this*  
2132 *document or the extent to which any license under such rights*  
2133 *might or might not be available; neither does it represent that it has*  
2134 *made any effort to identify any such rights. Information on OASIS's*  
2135 *procedures with respect to rights in OASIS specifications can be*  
2136 *found at the OASIS website. Copies of claims of rights made*  
2137 *available for publication and any assurances of licenses to be*  
2138 *made available, or the result of an attempt made to obtain a*  
2139 *general license or permission for the use of such proprietary rights*  
2140 *by implementors or users of this specification, can be obtained*  
2141 *from the OASIS Executive Director.*

2142 OASIS invites any interested party to bring to its attention any  
2143 copyrights, patents or patent applications, or other proprietary  
2144 rights which may cover technology that may be required to

2145 implement this specification. Please address the information to the  
2146 OASIS Executive Director.

2147 This document and translations of it may be copied and furnished  
2148 to others, and derivative works that comment on or otherwise  
2149 explain it or assist in its implementation may be prepared, copied,  
2150 published and distributed, in whole or in part, without restriction of  
2151 any kind, provided that the above copyright notice and this  
2152 paragraph are included on all such copies and derivative works.  
2153 However, this document itself may not be modified in any way,  
2154 such as by removing the copyright notice or references to OASIS,  
2155 except as needed for the purpose of developing OASIS  
2156 specifications, in which case the procedures for copyrights defined  
2157 in the OASIS Intellectual Property Rights document must be  
2158 followed, or as required to translate it into languages other than  
2159 English. The limited permissions granted above are perpetual and  
2160 will not be revoked by OASIS or its successors or assigns.

2161 This document and the information contained herein is provided on  
2162 an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES,  
2163 EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
2164 ANY WARRANTY THAT THE USE OF THE INFORMATION  
2165 HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
2166 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
2167 PARTICULAR PURPOSE.